Timex/Sinclair 2068 Personal Color Computer
Diagnostics Cartridge
Copyright 1985 By: Carl A. Miles
Version ART 1.0 - Nov 10,1985

OPERATION MANUAL
Version ART 1.0 - Nov 26,1985

Diagnostics Cartridge Version ART 1.0

By: Carl A. Miles
Copyright 1985

This is an LROS cartridge designed to test the Timex/Sinclair 2068 Personal Color Computer for defects in key hardware and software functions. In most cases, a simple PASS/FAIL result is given. However, in some cases, further testing will be needed on the part of the technician testing and repairing the computer. The tests are as follows:

1} Home RAM Test
2} CPU Test
3} SCLD Test
4} Home ROM Test
5} Extension ROM Test
6} Cassette Test
7} Keyboard Test
8} Test All Functions

The above tests appear in the Main Menu exactly as shown above. I will explain how each test is to be used and how the result of each test should be interpreted.

Some tests use the Home RAM as a temporary storage area for running programs and therefore will NOT function if the Home RAM test has FAILED or has not yet been run. These tests are as follows:

1} SCLD Test
2} Home ROM Test
3} Extension ROM Test

The Cassette Test uses very little Home RAM and can function without the Home RAM test's permission.

The CPU Test and Keyboard Test do NOT use any Home RAM to function. Therefore, these tests can be run regardless of if the Home RAM is good or bad.

THE TESTS

1} Home RAM Test

The Home RAM Test will test all 48K of the Home RAM. This test is run ten times for the sake of complete saftey. If the Home RAM can pass ten tests, then the odds are that the Home RAM is indeed good. If this test finds an error, it will report it and continue to the next 16K block of memory (the T/S 2068's Home RAM is split into three blocks of 16K each) and continue the test. If an error is found in the last 16K blobk, it will report it, lock the Home RAM dependent programs from use, and end. Check the chips given by the test in the order shown. The RAM chips should be COOL to the touch. If a RAM chip is HOT to the touch, the RAM chip is most likely BAD. Replace it. The other chips must be tested. The quickest method is to take the suspected part, put it into a new, good machine, and see if the new machine works correctly. If the new machine DOES work, the part is OK; however, if the new machine fails to work, the part is BAD. Replace it.

The Home RAM Test is very straightforward and does not require and further information.

2} CPU Test

The CPU Test will put the Z-80A microprocessor through its paces. Simply put, if this test passes, the CPU is OK; however, if this test FAILS, replace the CPU chip. Very straightforward.

3} SCLD Test

The SCLD Test will test the SCLD's bank-switching and system control ports. As with the CPU test, if this test passes, the SCLD can bank-switch and control the computer correctly. However, this is not an indication that the SCLD is 100% OK. The Home RAM Test, this test, the Cassette Test and the Keyboard Test must all pass for the SCLD to be CONSIDERED good. The SCLD is a very complex chip that also controls the video memory and screen. If the video output is OK, then the SCLD can be CONSIDERED good. Please note that there is no way you can say the SCLD is 100% good. If it passes all the tests and the video is good, the SCLD can be CONSIDERED good, but don't count on it.

If the SCLD test fails, then replace the SCLD chip. Very straightforward.

4} Home ROM Test

The Home ROM Test is also very simple. If the test passes, the Home ROM is good. If the test fails, replace the Home ROM chip.

5} Extension ROM Test

As with the Home ROM test, if this test passes, the EXROM is good. If this test fails, replace the EXROM chip. Very simple.

6} Cassette Test

This test will SAVE a 1K block of ROM to the tape five times. Set up the tape for a SAVE and follow the directions on the screen. After the test saves five times, the test will then verify the tape. Set up the tape for loading and follow the directions on the screen. If this test passes, all discrete components and the SCLD can be considered good. If this test fails, check all applicable discrete components and, if necessary, the SCLD input and output pins for the cassette and buzzer. Use CAPS SHIFT/BREAK To break out of this test. The test will then FAIL. If you have to end the test this way, the cassette test has indeed FAILED.

7} Keyboard Test

This test will test all 42 keys including the spacebar. It also tests the SHIFTED function keys. The order the keys need to be pressed are as follows:

1,2,3,4,5,6,7,8,9,0
q,w,e,r,t,y,u,i,o,p
a,s,d,f,g,h,j,k,l, ENTER KEY
CAPS SHIFT ,z,x,c,v,b,n,m, SYMBOL SHIFT, BREAK, CAPS SHIFT
SPACEBAR
CAPS SHIFT/5 (Cursor Left)
CAPS SHIFT/6 (Cursor Down)
CAPS SHIFT/7 (Cursor Up)
CAPS SHIFT/8 (Cursor Right)
SYMBOL SHIFT/m or '.' (decimal point, period)
SYMBOL SHIFT/v or '/' (backslash, divide symbol)
SYMBOL SHIFT/j or '-' (hypen, subtraction symbol)

If you pressed all keys in the order above, the test will pass and will tell you so. All keys will produce a short 'beep' when ACCEPTED. If a key does not produce a 'beep', it could be you did not follow the sequence correctly or the key is bad. If you followed the sequence incorrectly, press CAPS SHIFT/BREAK. The test will say the keyboard test failed. In fact, if you ever break out of the keyboard test this way, the test will indeed fail as YOU had to break out. Normally this does not occur. After the 'Press Any Key To Continue...' message, just rerun the keyboard test and follow the sequence to the letter!

If you have followed the sequence to the letter and a key still refuses to be accepted, the key has failed. Use the CAPS SHIFT/BREAK method described above to break out. If this is the case, the keyboard test has indeed FAILED. Check the physical keyboard itself and the diodes CR6 thru CR13. If these check out OK, then the SCLD chip is at fault. Replace the SCLD chip if this is the case. If the diodes are bad, replace the faulty ones and retest until the keyboard passes. If the physical keyboard itself is bad, either repair the keyboard or replace it. Then retest again.

8} Test All Functions

This is not itself a test. It simply sets up the program so the tests are performed in 'dasiy chain' or one after the other. This is simply a convience feature and should be used initially, with the individal tests only called after the faulty hardware/software have been identified and repaired.

When this feature is enabled and if it goes through ALL seven tests with all seven tests PASSING, the T/S 2068 under test can be considered good and should be retested one final time using this feature. If it passes all seven tests again, put that T/S 2068 into service.

IMPORTANT: Do not forget to record the serial number on the actual P.C. Board. Also record what was wrong with the computer under test before it was fixed so if the same computer comes back, you can see what was wrong and what was done to repair it.

Timex/Sinclair 2068 Personal Color Computer Diagnostics Cartridge
Copyright 1985 By: Carl A. Miles
Version ART 1.0 - Nov 10,1985

OPERATION MANUAL
Version ART 1.0 - Nov 26,1985

Timex/Sinclair 2068 & T/S 2068 are trademarks of Timex Computer Corporation
All Rights Reserved

Timex/Sinclair 2068 Color Computer
Technical Manual
For Diagnostics Cartridge Version ART 1.0 - Nov 10,1985
Copyright 1985 By: Carl A. Miles
Version ART 1.0 - Nov 25,1985

```
Header Code - Version ART 1.0 - By: Carl A. Miles - Nov 25,1985

org 00000h

start:
db      0ffh    ;not used
db      01h     ;LROS Present flag
db      13h     ;LSB of start address
db      00h     ;MSB of start address
db      0fch    ;chunk selection byte
db      0ffh    ;not used
db      0ffh    ;not used
db      0ffh    ;not used
rst 08h:
jp tape_fail    ;tape fail error call
db      0ffh    ;not used
db      0ffh    ;not used
db      0ffh    ;not used
db      0ffh    ;not used
db      0ffh    ;not used
rst 10h:
jp print_a      ;print restart
init:
di              ;disable interrupts
ld hl,(rmtp)    ;get RAMTOP
ld (hl),3eh     ;set ramtop for '3eh'
dec hl          ;set up stack
ld sp,hl        ;the stack set up now
ld hl,5cb6h     ;define new channel area
ld (chan),hl    ;define it now
ld de,15afh     ;source address of channel table
jr pass1        ;jump over restart 28h
db      0ffh    ;not used
db      0ffh    ;not used
rst 28h:
jp calc         ;calculator restart
pass1:
ld bc,0015h     ;length of channel table
ex de,hl        ;correct the registers
ldir            ;copy channel table into RAM
ld hl,15c6h     ;source address of stream table
jr pass2        ;jump over restart 38h
db      0ffh    ;not used
db      0ffh    ;not used
rst 38h:
ei              ;enable interrupts
ret             ;return to routine
pass2:
ld de,5c10h     ;address for stream table in RAM
ld bc,000eh     ;length of stream table
ldir            ;copy table into RAM
ld hl,5c92h     ;get calculator stack bottom...
ld (5c68),hl    ;and clear the calculator stack
```

```
Init Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 00048h

init:
ld a,03h                ;prepare border color for MAGENTA
call stack_a            ;stack color on stack
call border             ;set the border color now
ld a,(atrp)             ;get permanent attributes
rrca                    ;shift the paper...
rrca                    ;color to...
rrca                    ;bits 0 thru 2
and 0f8h                ;discard old color
add a,05h               ;enter new color (cyan)
rlca                    ;effect the...
rlca                    ;new paper...
rlca                    ;color
ld (atrp),a             ;effect the color now
ld a,0f8h               ;prepare to mask ink color
and (atrp)              ;do it now
ld (atrp),a             ;effect new color (black)
ld bc,01bah             ;length of title string
ld de,386eh             ;address of title string
call print_cls          ;clear screen and print title string
```

```
Keyboard Scan Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 00071h

key_scan:
ld bc,0f7fe             ;port for keys 1-5
in a,(c)                ;get keyboard status
cp 1eh                  ;is it a '1'?
jr z,ram_test           ;if yes, goto RAM test
cp 1dh                  ;is it a '2'?
jp z,cpu_test           ;if yes, goto cpu test
cp 1bh                  ;is it a '3'?
jp z,scld_test          ;if yes, goto scld test
cp 17h                  ;is it a '4'?
jp z,home rom_test      ;if yes, goto home rom test
cp 0fh                  ;is it a '5'?
jp z,exrom_test         ;if yes, goto exrom test
ld bc,0effeh            ;port for keys 6-0
in a,(c)                ;get keyboard status
cp 0fh                  ;is it a '6'?
jp z,cass_test          ;if yes, goto cassette test
cp 17h                  ;is it a '7'?
jp z,key_test           ;if yes, goto keyboard test
cp 1bh                  ;is it a '8'?
jp z,auto_flag          ;if yes, goto auto_flag routine
jr key_scan             ;no key pressed, rescan forever...
```

```
String Printing Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 00a4h

print_cls:
push bc                 ;save bc
push de                 ;save de
ld a,02h                ;prepare to open screen channel
call chan_open          ;open the channel
call cls                ;clear the screen
jr print1               ;jump over string code
string:
push bc                 ;save bc
push de                 ;save de
ld a,02h                ;prepare to open screen channel
call chan_open          ;open the channel
pop de                  ;restore address of string
pop bc                  ;restore length of string
call print_string       ;print string
ret                     ;return to calling routine
```

```
Beeper Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 000bdh

pass_beep:
ld a,02h                ;prepare a beep of...
call stack_a            ;two seconds...
ld a,1eh                ;and a pitch of...
beep_short:
call stack_a            ;thirty
call beep               ;produce the beep
ret                     ;return to calling routine
fail_beep:
ld a,02                 ;prepare a beep of...
call stack_a            ;two seconds...
xor a                   ;and a pitch of zero
jr beep_short           ;jump back to common code
```

```
Wait Key Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 000d3h

wait_key:
ei                      ;enable interrupts
wait_loop:
ld b,3ch                ;set up a counter of 100 milliseconds
halt                    ;halt processor for 1 millisecond
djnz,wait_loop          ;if loop not complete, reloop
di                      ;disable interrupts
no_key:
call keyscan            ;call ROM keyboard scan routine
inc e                   ;get key pressed
xor a                   ;clear a
cp e                    ;is no key pressed?
je z,no_key             ;if so, loop until key pressed
ret                     ;return to calling routine
```

```
Auto Flag Check Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 000e3h

auto_check:
ld a,(5cb0)             ;get auto flag
and a                   ;effect zero flag
ret                     ;return to calling routine
```

```
Continue String Code - Version ART 1.0 - By: Carl A. Miles - Nov 26,1985

org 00e8h

cont:
ld bc,001eh             ;length of continue string
ld de,1577h             ;address of continue string
call string             ;print the string
ret                     ;return to calling routine
```

8K, 16K AND 32K EPROM PROGRAMMING

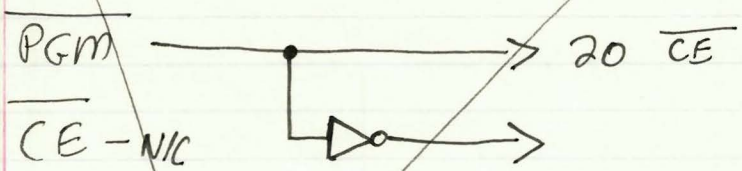
8K AND 16K as per normal

32K:

Change this:

$\overline{PGM}$ becomes $\overline{CE}$
(27) ← pins → (20)

→ Invert this and goto switch



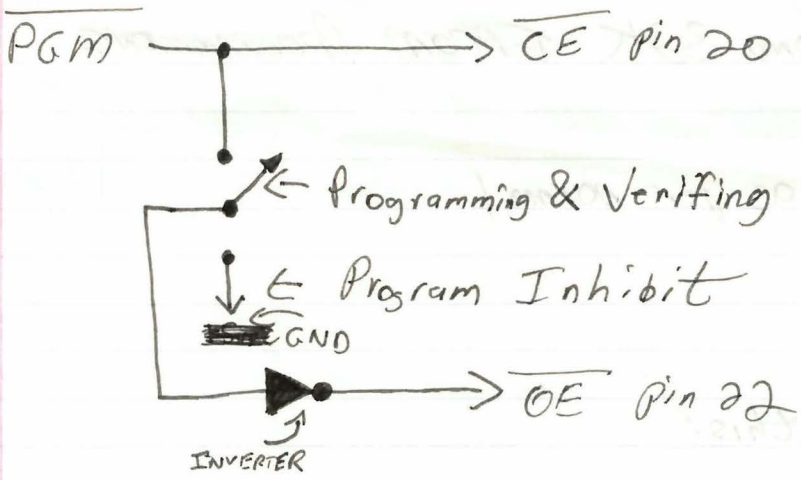
$\overline{CE}$ – N/C

$\overline{OE}$ – N/C

V_{PP} = 12.5 VDC!!
NOT 21V!!

V_{CC} = 5 VDC

GND = Ø VDC



$\overline{CE}$ - N/C

$\overline{OE}$ - N/C

VPP = 12.5 V NOT 21 V.!!

U1 - SWITCHING +5VOLT REGULATOR

	PIN	VOM READING	LOGIC PROBE READING
SIDE 1:	1	Ø V	
	2	Ø V	
	3	Ø V (GND)	
	4	.2V	
	5	19V	
	6	Ø V	
	7	Ø V	
	8	1.2V	
SIDE 2:	9	1.2V	
	10	1.2V	
	11	Ø V	
	12	.75V	
	13	19V (Vcc)	
	14	19V	
	15	15.5V	
	16	15.5V	

NOTES: VERSION A BOARD
10Ω RESISTOR IN-LINE WITH +5VDC POWER SUPPLY TO THE SCLD CHIP

U3 STANDARD CELL LOGIC DESIGN (SCLD) (BBB)

	PIN	VOM READING	LOGIC PROBE READING
SIDE 1:	1	2v	
	2	1.8v	
	3	1.65v	
	4	1.5v	
	5	2.55v	
	6	2.1v	
	7	2.4v	
	8	1.8v	
	9	1.7v	
SIDE 2:	10	2.5v	
	11	4.65v	
	12	3v (ALSO TO US BOARD)	
	13	2.25v	
	14	.2v	
	15	.5v	
	16	.25v	
	17	Øv	
	18	Øv	
	19	2.75v	
	20	1v (ALSO TO US BOARD)	
	21	3.3v	
	22	2.4v	
	23	1.7v	
	24	2.7v	
	25	1.7v	
	26	Øv (GND)	
SIDE 3:	27	1.5v	
	28	2.7v	
	29	2.3v	
	30	1.9v	

NOTES: VERSION A BOARD
10 Ω RESISTOR IN-LINE WITH +5VDC POWER SUPPLY TO THE SCLD CHIP

U3 STANDARD CELL LOGIC DESIGN (SCLO) (BBB)

	PIN	VOM READING	LOGIC PROBE READING
SIDE3:	31	1.2v	
	32	4.55v (ALSO TO uC BOARD)	
	33	1.6v	
	34	3.8v	
	35	3.75v	
	36	3.35v	
	37	4.6v	
	38	2.8v	
	39	4.65v	
	40	4.65v	
	41	4v	
	42	3.1v	
	43	4.8v (Vcc) 10Ω IN-LINE	
SIDE4:	44	1.15v	
	45	.05v	
	46	3.6v	
	47	3.6v	
	48	3.6v	
	49	3.15v	
	50	3.15v	
	51	3.15v	
	52	3.15v	
	53	3.15v	
	54	4.35v	
	55	2v	
	56	2.5v	
	57	1.8v	
	58	3.0v	
	59	2.4v	
	60	2.1v	

NOTES: VERSION A BOARD
10Ω RESISTOR IN-LINE WITH +5VDC POWER SUPPLY TO THE SCLO CHIP

U3 STANDARD CELL LOGIC DESIGN (SCLD) (BBB)

	PIN	VOM READING	LOGIC PROBE READING
SIDE 1:	61	2.75v	
	62	1.8v	
	63	2.3v	
	64	3.65v	
	65	3.6v	
	66	1.8v (ALSO TO US BOARD)	
	67	4.65v	
	68	2.9v	

NOTES: VERSION A BOARD
10 Ω RESISTOR IN-LINE WITH +5VDC POWER SUPPLY TO THE SCLD CHIP.

```
   1 RANDOMIZE USR 60000: LET A=
PEEK 60162
   2 IF a<>0 THEN PRINT "Header(
ess File": GO TO 10
   3 PRINT "This File Read In"
   4 LET a=PEEK 60064
   5 IF a=0 THEN PRINT "Basic Pr
ogram"
   6 IF a=3 THEN PRINT "Machine
Code Program"
  10 PRINT "Next Header...": PAU
SE 0: RUN
```

```
EA60 F3              DI
EA61 DBFF            IN A,(FF)
EA63 CBFF            SET 7,A
EA65 D3FF            OUT (FF),A
EA67 3E01            LD A,01
EA69 D3F4            OUT (F4),A
EA6B DD21A0EA        LD IX,EAA0
EA6F 111100          LD DE,0011
EA72 AF              XOR A
EA73 37              SCF
EA74 CDFC00          CALL 00FC
EA77 F3              DI
EA78 F5              PUSH AF
EA79 AF              XOR A
EA7A D3F4            OUT (F4),A
EA7C DBFF            IN A,(FF)
EA7E CBBF            RES 7,A
EA80 D3FF            OUT (FF),A
EA82 FB              EI
EA83 F1              POP AF
EA84 3200EB          LD (EB00),A
EA87 7A              LD A,D
EA88 3201EB          LD (EB01),A
EA8B 7B              LD A,E
EA8C 3202EB          LD (EB02),A
EA8F DDE5            PUSH IX
EA91 E1              POP HL
EA92 7C              LD A,H
EA93 D7              RST 10H
EA94 7D              LD A,L
EA95 D7              RST 10H
EA96 C9              RET
```

ROM CHECKSUM RENDER

```
EA60 F3              DI
EA61 010000          LD BC,0000
EA64 110000          LD DE,0000
EA67 210000          LD HL,0000
EA6A C5        RETU  PUSH BC
EA6B 010000          LD BC,0000
EA6E 1A        LOOP  LD A,(DE)
EA6F 4F              LD C,A
EA70 AF              XOR A
EA71 ED4A            ADC HL,BC
EA73 380D            JR C,OVER
EA75 13              INC DE
EA76 7A              LD A,D
EA77 FE40            CP 40
EA79 20F3            JR NZ,LOOP
EA7B 7D              LD A,L
EA7C D7              RST 10H
EA7D 7C              LD A,H
EA7E D7              RST 10H
EA7F C1              POP BC
EA80 FB              EI
EA81 C9              RET
EA82 C1        OVER  POP BC
EA83 03              INC BC
EA84 13              INC DE
EA85 18E3            JR RETU
```

ADDRESS

LOW BYTE

HIGH BYTE

UPPER 16 BITS

BACK

MNEMON	OCT	HEX	STATES	DESCRIPTION
SPHL	371	F9	5	Load SP from H&L
SHLD aa	042	22	16	Store H&L at LOC aa
STA aa	062	32	13	Store A at LOC aa
STAX B	002	02	7	Store A at LOC(B&C)
STAX D	022	12	7	Store A at LOC(D&E)
STC	067	37	4	Set carry
SUB A	227	97	4	Clear A and flags
SUB B	220	90	4	Subtract B from A
SUB C	221	91	4	Subtract C from A
SUB D	222	92	4	Subtract D from A
SUB E	223	93	4	Subtract E from A
SUB H	224	94	4	Subtract H from A
SUB L	225	95	4	Subtract L from A
SUB M	226	96	7	Subtract LOC(H&L) from A
SUI v	326	D6	7	Subtract v from A
SBB A	237	9F	4	Set A to minus carry
SBB B	230	98	4	Subtract B and borrow from A
SBB C	231	99	4	Subtract C and borrow from A
SBB D	232	9A	4	Subtract D and borrow from A
SBB E	233	9B	4	Subtract E and borrow from A
SBB H	234	9C	4	Subtract H and borrow from A
SBB L	235	9D	4	Subtract L and borrow from A
SBB M	236	9E	7	Subtract LOC(H&L) and borrow from A
SBI v	336	DE	7	Subtract v and borrow from A
XCHG	353	EB	4	Exchange D&E with H&L
XTHL	343	E3	18	Exchange top of stack with H&L
XRA A	257	AF	4	Clear A and flags
XRA B	250	A8	4	Exclusive OR B with A
XRA C	251	A9	4	Exclusive OR C with A
XRA D	252	AA	4	Exclusive OR D with A
XRA E	253	AB	4	Exclusive OR E with A
XRA H	254	AC	4	Exclusive OR H with A
XRA L	255	AD	4	Exclusive OR L with A
XRA M	256	AE	7	Exclusive OR LOC(H&L) with A
XRI v	356	EE	7	Exclusive OR v with A

STATUS FLAGS

flag	description	flag=1	flag=0
Z	Result is Zero	Zero(Z)	Not Zero(NZ)
S	Sign of Result(MSB)	Minus(M)	Plus(P)
P	Parity of Result	Even(PE)	Odd(PO)
C	Carry from Result	Carry(C)	No Carry(NC)
AC	Auxiliary Carry from bit 4 of Result	Carry	No Carry

INSTRUCTIONS AFFECTING FLAGS

ACI all flags	ORA all flags,C=AC=0
ADC all flags	ORI all flags,C=AC=0
ADD all flags	POP PSW all flags
ADI all flags	RAL only C flag
ANA all flags,C=AC=0	RAR only C flag
ANI all flags,C=AC=0	RLC only C flag
CMC only C flag,C=1-C	RRC only C flag
CMP all flags	SBB all flags
CPI all flags	SBI all flags
DAA all flags	STC only C flag,C=1
DAD only C flag	SUB all flags
DCR all but C flag	SUI all flags
INR all but C flag	XRA all flags,C=AC=0
	XRI all flags,C=AC=0

COMPARE INSTRUCTION

Result of CMP B instruction

A < B Carry	A ≥ B No Carry
A = B Zero	A ≠ B Not Zero
A > B Not Zero And No Carry	A ≤ B Zero or Carry

INSTRUCTION FORMATS

ONE BYTE INSTRUCTIONS

OP-CODE

TWO BYTE INSTRUCTIONS

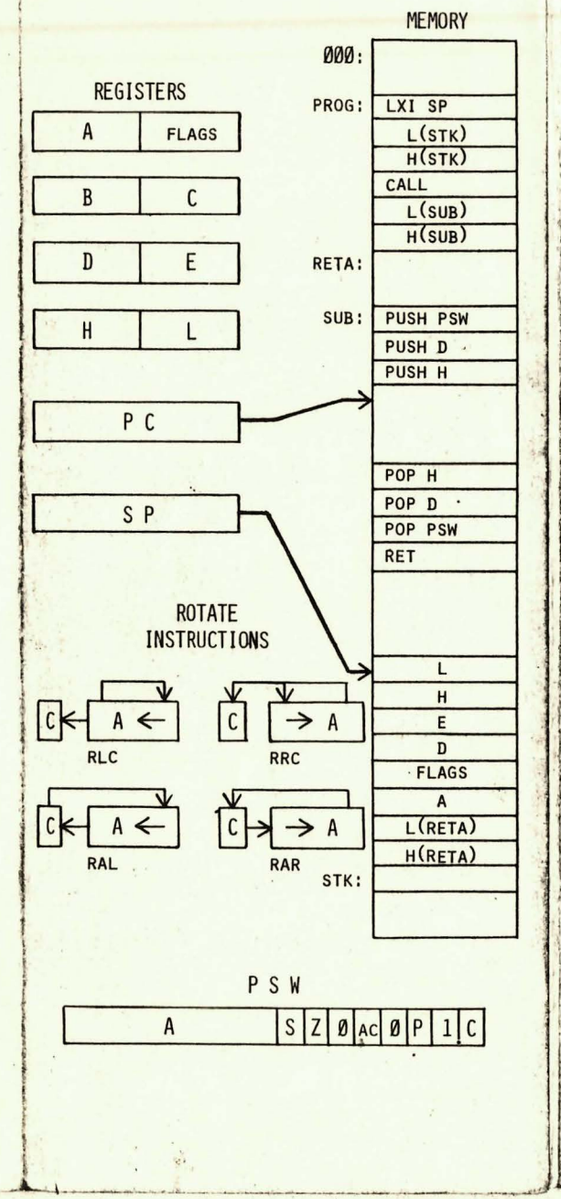
OP-CODE	v

THREE BYTE INSTRUCTIONS

OP-CODE	L(vv)	H(vv)

OP-CODE	L(aa)	H(aa)

INTERNAL STRUCTURE



reference card for the intel 8080*

< ascii mnemonics >

< notation flags >

< conversions structure >

*Trademark INTEL Corp.

8080 CARD, Copyright 1975, Don Mimlitch

made in U.S.A.

BACK

MNEMON	OCT	HEX	STATES	DESCRIPTION
ADD A	207	87	4	Double A (shift A left)
ADD B	200	80	4	Add B to A
ADD C	201	81	4	Add C to A
ADD D	202	82	4	Add D to A
ADD E	203	83	4	Add E to A
ADD H	204	84	4	Add H to A
ADD L	205	85	4	Add L to A
ADD M	206	86	7	Add LOC(H&L) to A
ADI v	306	C6	7	Add v to A
ADC A	217	8F	4	Double A with carry (shift left with carry)
ADC B	210	88	4	Add B to A with carry
ADC C	211	89	4	Add C to A with carry
ADC D	212	8A	4	Add D to A with carry
ADC E	213	8B	4	Add E to A with carry
ADC H	214	8C	4	Add H to A with carry
ADC L	215	8D	4	Add L to A with carry
ADC M	216	8E	7	Add LOC(H&L) to A with carry
ACI v	316	CE	7	Add v to A with carry
ANA A	247	A7	4	Test A and clear carry
ANA B	240	A0	4	AND B with A
ANA C	241	A1	4	AND C with A
ANA D	242	A2	4	AND D with A
ANA E	243	A3	4	AND E with A
ANA H	244	A4	4	AND H with A
ANA L	245	A5	4	AND L with A
ANA M	246	A6	7	AND LOC(H&L) with A
ANI v	346	E6	7	AND v with A
CALL aa	315	CD	17	Call subroutine aa
CZ aa	314	CC	11/17	If zero CALL
CNZ aa	304	C4	11/17	If not zero CALL
CP aa	364	F4	11/17	If plus CALL
CM aa	374	FC	11/17	If minus CALL
CC aa	334	DC	11/17	If carry CALL
CNC aa	324	D4	11/17	If no carry CALL
CPE aa	354	EC	11/17	If even parity CALL
CPO aa	344	E4	11/17	If odd parity CALL
CMA	057	2F	4	Compliment A
CMC	077	3F	4	Compliment carry
CMP A	277	BF	4	Clear flags
CMP B	270	B8	4	Compare A with B
CMP C	271	B9	4	Compare A with C
CMP D	272	BA	4	Compare A with D
CMP E	273	BB	4	Compare A with E
CMP H	274	BC	4	Compare A with H
CMP L	275	BD	4	Compare A with L
CMP M	276	BE	7	Compare A with LOC(H&L)
CPI v	376	FE	7	Compare A with v

MNEMON	OCT	HEX	STATES	DESCRIPTION
DAA	047	27	4	Decimal adjust A
DAD B	011	09	10	Add B&C to H&L
DAD D	031	19	10	Add D&E to H&L
DAD H	051	29	18	Double H&L (shift H&L left)
DAD SP	071	39	10	Add SP to H&L
DCR A	075	3D	5	Decrement A
DCR B	005	05	5	Decrement B
DCR C	015	0D	5	Decrement C
DCR D	025	15	5	Decrement D
DCR E	035	1D	5	Decrement E
DCR H	045	25	5	Decrement H
DCR L	055	2D	5	Decrement L
DCR M	065	35	10	Decrement LOC(H&L)
DCX B	013	0B	5	Decrement B&C
DCX D	033	1B	5	Decrement D&E
DCX H	053	2B	5	Decrement H&L
DCX SP	073	3B	5	Decrement SP
DI	363	F3	4	Disable interrupts
EI	373	FB	4	Enable interrupts
HLT	166	76	7	halt until interrupt
IN v	333	DB	10	Input from device v
INR A	074	3C	5	Increment A
INR B	004	04	5	Increment B
INR C	014	0C	5	Increment C
INR D	024	14	5	Increment D
INR E	034	1C	5	Increment E
INR H	044	24	5	Increment H
INR L	054	2C	5	Increment L
INR M	064	34	10	Increment LOC(H&L)
INX B	003	03	5	Increment B&C
INX D	023	13	5	Increment D&E
INX H	043	23	5	Increment H&L
INX SP	063	33	5	Increment SP
JMP aa	303	C3	10	Jump to LOC aa
JZ aa	312	CA	10	If zero JMP
JNZ aa	302	C2	10	If not zero JMP
JP aa	362	F2	10	If plus JMP
JM aa	372	FA	10	If minus JMP
JC aa	332	DA	10	If carry JMP
JNC aa	322	D2	10	If no carry JMP
JPE aa	352	EA	10	If even parity JMP
JPO aa	342	E2	10	If odd parity JMP
LDA aa	072	3A	13	Load A from LOC aa
LDAX B	012	0A	7	Load A from LOC(B&C)
LDAX D	032	1A	7	Load A from LOC(D&E)
LHLD aa	052	2A	16	Load H&L from LOC aa
LXI B,vv	001	01	10	Load B&C with vv
LXI D,vv	021	11	10	Load D&E with vv
LXI H,vv	041	21	10	Load H&L with vv
LXI SP,vv	061	31	10	Load SP with vv

MNEMON	OCT	HEX	STATES	DESCRIPTION
MOV A,B	170	78	5	Move B to A
MOV A,C	171	79	5	Move C to A
MOV A,D	172	7A	5	Move D to A
MOV A,E	173	7B	5	Move E to A
MOV A,H	174	7C	5	Move H to A
MOV A,L	175	7D	5	Move L to A
MOV A,M	176	7E	7	Move LOC(H&L) to A
MOV B,A	107	47	5	Move A to B
MOV B,C	101	41	5	Move C to B
MOV B,D	102	42	5	Move D to B
MOV B,E	103	43	5	Move E to B
MOV B,H	104	44	5	Move H to B
MOV B,L	105	45	5	Move L to B
MOV B,M	106	46	7	Move LOC(H&L) to B
MOV C,A	117	4F	5	Move A to C
MOV C,B	110	48	5	Move B to C
MOV C,D	112	4A	5	Move D to C
MOV C,E	113	4B	5	Move E to C
MOV C,H	114	4C	5	Move H to C
MOV C,L	115	4D	5	Move L to C
MOV C,M	116	4E	7	Move LOC(H&L) to C
MOV D,A	127	57	5	Move A to D
MOV D,B	120	50	5	Move B to D
MOV D,C	121	51	5	Move C to D
MOV D,E	123	53	5	Move E to D
MOV D,H	124	54	5	Move H to D
MOV D,L	125	55	5	Move L to D
MOV D,M	126	56	7	Move LOC(H&L) to D
MOV E,A	137	5F	5	Move A to E
MOV E,B	130	58	5	Move B to E
MOV E,C	131	59	5	Move C to E
MOV E,D	132	5A	5	Move D to E
MOV E,H	134	5C	5	Move H to E
MOV E,L	135	5D	5	Move L to E
MOV E,M	136	5E	7	Move LOC(H&L) to E
MOV H,A	147	67	5	Move A to H
MOV H,B	140	60	5	Move B to H
MOV H,C	141	61	5	Move C to H
MOV H,D	142	62	5	Move D to H
MOV H,E	143	63	5	Move E to H
MOV H,L	145	65	5	Move L to H
MOV H,M	146	66	7	Move LOC(H&L) to H
MOV L,A	157	6F	5	Move A to L
MOV L,B	150	68	5	Move B to L
MOV L,C	151	69	5	Move C to L
MOV L,D	152	6A	5	Move D to L
MOV L,E	153	6B	5	Move E to L
MOV L,H	154	6C	5	Move H to L
MOV L,M	156	6E	7	Move LOC(H&L) to L
MOV M,A	167	77	7	Move A to LOC(H&L)
MOV M,B	160	70	7	Move B to LOC(H&L)
MOV M,C	161	71	7	Move C to LOC(H&L)
MOV M,D	162	72	7	Move D to LOC(H&L)
MOV M,E	163	73	7	Move E to LOC(H&L)
MOV M,H	164	74	7	Move H to LOC(H&L)
MOV M,L	165	75	7	Move L to LOC(H&L)

MNEMON	OCT	HEX	STATES	DESCRIPTION
MVI A,v	076	3E	7	Move v to A
MVI B,v	006	06	7	Move v to B
MVI C,v	016	0E	7	Move v to C
MVI D,v	026	16	7	Move v to D
MVI E,v	036	1E	7	Move v to E
MVI H,v	046	26	7	Move v to H
MVI L,v	056	2E	7	Move v to L
MVI M,v	066	36	10	Move v to LOC(H&L)
NOP	000	00	4	No operation
ORA A	267	B7	4	Test A and clear carry
ORA B	260	B0	4	OR B with A
ORA C	261	B1	4	OR C with A
ORA D	262	B2	4	OR D with A
ORA E	263	B3	4	OR E with A
ORA H	264	B4	4	OR H with A
ORA L	265	B5	4	OR L with A
ORA M	266	B6	7	OR LOC(H&L) with A
ORI v	366	F6	7	OR v with A
OUT v	323	D3	10	Output to device v
PCHL	351	E9	5	Jump to LOC(H&L)
POP B	301	C1	10	Pop B&C from stack
POP D	321	D1	10	Pop D&E from stack
POP H	341	E1	10	Pop H&L from stack
POP PSW	361	F1	10	Pop A and flags from stack
PUSH B	305	C5	11	Push B&C onto stack
PUSH D	325	D5	11	Push D&E onto stack
PUSH H	345	E5	11	Push H&L onto stack
PUSH PSW	365	F5	11	Push A and flags onto stack
RAL	027	17	4	Rotate A left through carry
RAR	037	1F	4	Rotate A right through carry
RLC	007	07	4	Rotate A left
RRC	017	0F	4	Rotate A right
RET	311	C9	10	Return from subroutine
RZ	310	C8	5/11	If zero RET
RNZ	300	C0	5/11	If not zero RET
RP	360	F0	5/11	If plus RET
RM	370	F8	5/11	If minus RET
RC	330	D8	5/11	If carry RET
RNC	320	D0	5/11	If no carry RET
RPE	350	E8	5/11	If even parity RET
RPO	340	E0	5/11	If odd parity RET
RST 0	307	C7	11	Restart subroutine 000
RST 1	317	CF	11	Restart subroutine 010
RST 2	327	D7	11	Restart subroutine 020
RST 3	337	DF	11	Restart subroutine 030
RST 4	347	E7	11	Restart subroutine 040
RST 5	357	EF	11	Restart subroutine 050
RST 6	367	F7	11	Restart subroutine 060
RST 7	377	FF	11	Restart subroutine 070

FRONT

HEXADECIMAL CONVERSION TABLE

1	1048576	65536	4096	256	16	1	1
2	2097152	131072	8192	512	32	2	2
3	3145728	196608	12288	768	48	3	3
4	4194304	262144	16384	1024	64	4	4
5	5242880	327680	20480	1280	80	5	5
6	6291456	393216	24576	1536	96	6	6
7	7340032	458752	28672	1792	112	7	7
8	8388608	524288	32768	2048	128	8	8
9	9437184	589824	36864	2304	144	9	9
A	10485760	655360	40960	2560	160	10	A
B	11534336	720896	45056	2816	176	11	B
C	12582912	786432	49152	3072	192	12	C
D	13631488	851968	53248	3328	208	13	D
E	14680064	917504	57344	3584	224	14	E
F	15728640	983040	61440	3840	240	15	F
	6	5	4	3	2	1	

POWERS OF SIXTEEN

1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15
18 446 744 073 709 551 616	16

POWERS OF EIGHT

1	0
8	1
64	2
512	3
4 096	4
32 768	5
262 144	6
2 097 152	7
16 777 216	8
134 217 728	9
1 073 741 824	10
8 589 934 592	11
68 719 476 736	12
549 755 813 888	13
4 398 046 511 104	14
35 184 372 088 832	15
281 474 976 710 656	16

POWERS OF TWO

1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24
33 554 432	25
67 108 864	26
134 217 728	27
268 435 456	28
536 870 912	29
1 073 741 824	30
2 147 483 648	31
4 294 967 296	32
8 589 934 592	33
17 179 869 184	34
34 359 738 368	35
68 719 476 736	36

OCTAL CONVERSION TABLE

1	32768	4096	512	64	8	1	1
2	65536	8192	1024	128	16	2	2
3	98304	12288	1536	192	24	3	3
4	131072	16384	2048	256	32	4	4
5	163840	20480	2560	320	40	5	5
6	196608	24576	3072	384	48	6	6
7	229376	28672	3584	448	56	7	7
	6	5	4	3	2	1	

NOTATION

Symbol	Description
A	Accumulator (8-bits)
B,C,D,E,H,L	Work Registers (8-bits)
B&C,D&E,H&L	Register Pairs (16-bits)
M	Contents of memory location pointed to by H&L
PC	Program Counter
SP	Stack Pointer
PSW	Processor Status Word
rp	Register Pair (B&C,D&E or H&L)
v	8 bit value after instruction
vv	16 bit value after instruction
aa	16 bit address after instruction
H(vv)	Most significant byte of vv
L(vv)	Least significant byte of vv
LOC aa	Contents of memory location aa
LOC(rp)	Contents of memory location pointed to by register pair rp

ASCII CONTROL CHARACTERS

char	key	description
ACK	CTL-F	Acknowledge
BEL	CTL-G	Bell
BS	CTL-H	Backspace
CAN	CTL-X	Cancel
CR	CTL-M	Carriage Return
DC1	CTL-Q	Device Control 1 (X-ON)
DC2	CTL-R	Device Control 2
DC3	CTL-S	Device Control 3 (X-OFF)
DC4	CTL-T	Device Control 4
DEL		Delete
DLE	CTL-P	Data Link Escape
EM	CTL-Y	End of Medium
ENQ	CTL-E	Enquiry
EOT	CTL-D	End of Transmission
ESC		Escape
ETB	CTL-W	End of Transmission Block
ETX	CTL-C	End of Text
FF	CTL-L	Form Feed
FS		File Separator
GS		Group Separator
HT	CTL-I	Horizontal Tab
LF	CTL-J	Line Feed
NAK	CTL-U	Negative Acknowledge
NUL		Null
RS		Record Separator
SI	CTL-O	Shift In
SO	CTL-N	Shift Out
SOH	CTL-A	Start of Heading
SP		Space (a blank)
STX	CTL-B	Start of Text
SUB	CTL-Z	Substitute
SYN	CTL-V	Synchronous Idle
US		Unit Separator
VT	CTL-K	Vertical Tab

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC		CYCLES	OCT
0	00	000	NUL	EVEN	NOP		1	000
1	01	001	SOH	ODD	LXI	B,vv	3	001
2	02	002	STX	ODD	STAX	B	2	002
3	03	003	ETX	EVEN	INX	B	1	003
4	04	004	EOT	ODD	INR	B	1	004
5	05	005	ENQ	EVEN	DCR	B	1	005
6	06	006	ACK	EVEN	MVI	B,v	2	006
7	07	007	BEL	ODD	RLC		1	007
8	08	010	BS	ODD				010
9	09	011	HT	EVEN	DAD	B	3	011
10	0A	012	LF	EVEN	LDAX	B	2	012
11	0B	013	VT	ODD	DCX	B	1	013
12	0C	014	FF	EVEN	INR	C	1	014
13	0D	015	CR	ODD	DCR	C	1	015
14	0E	016	SO	ODD	MVI	C,v	2	016
15	0F	017	SI	EVEN	RRC		1	017
16	10	020	DLE	ODD				020
17	11	021	DC1	EVEN	LXI	D,vv	3	021
18	12	022	DC2	EVEN	STAX	D	2	022
19	13	023	DC3	ODD	INX	D	1	023
20	14	024	DC4	EVEN	INR	D	1	024
21	15	025	NAK	ODD	DCR	D	1	025
22	16	026	SYN	ODD	MVI	D,v	2	026
23	17	027	ETB	EVEN	RAL		1	027
24	18	030	CAN	EVEN				030
25	19	031	EM	ODD	DAD	D	3	031
26	1A	032	SUB	ODD	LDAX	D	2	032
27	1B	033	ESC	EVEN	DCX	D	1	033
28	1C	034	FS	ODD	INR	E	1	034
29	1D	035	GS	EVEN	DCR	E	1	035
30	1E	036	RS	EVEN	MVI	E,v	2	036
31	1F	037	US	ODD	RAR		1	037
32	20	040	SP	ODD				040
33	21	041	!	EVEN	LXI	H,vv	3	041
34	22	042	"	EVEN	SHLD	aa	5	042
35	23	043	#	ODD	INX	H	1	043
36	24	044	$	EVEN	INR	H	1	044
37	25	045	%	ODD	DCR	H	1	045
38	26	046	&	ODD	MVI	H,v	2	046
39	27	047	'	EVEN	DAA		1	047
40	28	050	(	EVEN				050
41	29	051	)	ODD	DAD	H	3	051
42	2A	052	*	ODD	LHLD	aa	5	052
43	2B	053	+	EVEN	DCX	H	1	053
44	2C	054	,	ODD	INR	L	1	054
45	2D	055	-	EVEN	DCR	L	1	055
46	2E	056	.	EVEN	MVI	L,v	2	056
47	2F	057	/	ODD	CMA		1	057
48	30	060	0	EVEN				060
49	31	061	1	ODD	LXI	SP,vv	3	061
50	32	062	2	ODD	STA	aa	4	062
51	33	063	3	EVEN	INX	SP	1	063
52	34	064	4	ODD	INR	M	3	064
53	35	065	5	EVEN	DCR	M	3	065
54	36	066	6	EVEN	MVI	M,v	3	066
55	37	067	7	ODD	STC		1	067
56	38	070	8	ODD				070
57	39	071	9	EVEN	DAD	SP	3	071
58	3A	072	:	EVEN	LDA	aa	4	072
59	3B	073	;	ODD	DCX	SP	1	073
60	3C	074	<	EVEN	INR	A	1	074
61	3D	075	=	ODD	DCR	A	1	075
62	3E	076	>	ODD	MVI	A,v	2	076
63	3F	077	?	EVEN	CMC		1	077

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC		CYCLES	OCT
64	40	100	@	ODD	MOV	B,B	1	100
65	41	101	A	EVEN	MOV	B,C	1	101
66	42	102	B	EVEN	MOV	B,D	1	102
67	43	103	C	ODD	MOV	B,E	1	103
68	44	104	D	EVEN	MOV	B,H	1	104
69	45	105	E	ODD	MOV	B,L	1	105
70	46	106	F	ODD	MOV	B,M	2	106
71	47	107	G	EVEN	MOV	B,A	1	107
72	48	110	H	EVEN	MOV	C,B	1	110
73	49	111	I	ODD	MOV	C,C	1	111
74	4A	112	J	ODD	MOV	C,D	1	112
75	4B	113	K	EVEN	MOV	C,E	1	113
76	4C	114	L	ODD	MOV	C,H	1	114
77	4D	115	M	EVEN	MOV	C,L	1	115
78	4E	116	N	EVEN	MOV	C,M	2	116
79	4F	117	O	ODD	MOV	C,A	1	117
80	50	120	P	EVEN	MOV	D,B	1	120
81	51	121	Q	ODD	MOV	D,C	1	121
82	52	122	R	ODD	MOV	D,D	1	122
83	53	123	S	EVEN	MOV	D,E	1	123
84	54	124	T	ODD	MOV	D,H	1	124
85	55	125	U	EVEN	MOV	D,L	1	125
86	56	126	V	EVEN	MOV	D,M	2	126
87	57	127	W	ODD	MOV	D,A	1	127
88	58	130	X	ODD	MOV	E,B	1	130
89	59	131	Y	EVEN	MOV	E,C	1	131
90	5A	132	Z	EVEN	MOV	E,D	1	132
91	5B	133	[	ODD	MOV	E,E	1	133
92	5C	134	\	EVEN	MOV	E,H	1	134
93	5D	135	]	ODD	MOV	E,L	1	135
94	5E	136	↑	ODD	MOV	E,M	2	136
95	5F	137	←	EVEN	MOV	E,A	1	137
96	60	140	`	EVEN	MOV	H,B	1	140
97	61	141	a	ODD	MOV	H,C	1	141
98	62	142	b	ODD	MOV	H,D	1	142
99	63	143	c	EVEN	MOV	H,E	1	143
100	64	144	d	ODD	MOV	H,H	1	144
101	65	145	e	EVEN	MOV	H,L	1	145
102	66	146	f	EVEN	MOV	H,M	2	146
103	67	147	g	ODD	MOV	H,A	1	147
104	68	150	h	ODD	MOV	L,B	1	150
105	69	151	i	EVEN	MOV	L,C	1	151
106	6A	152	j	EVEN	MOV	L,D	1	152
107	6B	153	k	ODD	MOV	L,E	1	153
108	6C	154	l	EVEN	MOV	L,H	1	154
109	6D	155	m	ODD	MOV	L,L	1	155
110	6E	156	n	ODD	MOV	L,M	2	156
111	6F	157	o	EVEN	MOV	L,A	1	157
112	70	160	p	ODD	MOV	M,B	2	160
113	71	161	q	EVEN	MOV	M,C	2	161
114	72	162	r	EVEN	MOV	M,D	2	162
115	73	163	s	ODD	MOV	M,E	2	163
116	74	164	t	EVEN	MOV	M,H	2	164
117	75	165	u	ODD	MOV	M,L	2	165
118	76	166	v	ODD	HLT		2	166
119	77	167	w	EVEN	MOV	M,A	2	167
120	78	170	x	EVEN	MOV	A,B	1	170
121	79	171	y	ODD	MOV	A,C	1	171
122	7A	172	z	ODD	MOV	A,D	1	172
123	7B	173	{	EVEN	MOV	A,E	1	173
124	7C	174	|	ODD	MOV	A,H	1	174
125	7D	175	}	EVEN	MOV	A,L	1	175
126	7E	176	~	EVEN	MOV	A,M	2	176
127	7F	177	DEL	ODD	MOV	A,A	1	177

FRONT

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
128	80	200	NUL	ODD	ADD B	1	200
129	81	201	SOH	EVEN	ADD C	1	201
130	82	202	STX	EVEN	ADD D	1	202
131	83	203	ETX	ODD	ADD E	1	203
132	84	204	EOT	EVEN	ADD H	1	204
133	85	205	ENQ	ODD	ADD L	1	205
134	86	206	ACK	ODD	ADD M	2	206
135	87	207	BEL	EVEN	ADD A	1	207
136	88	210	BS	EVEN	ADC B	1	210
137	89	211	HT	ODD	ADC C	1	211
138	8A	212	LF	ODD	ADC D	1	212
139	8B	213	VT	EVEN	ADC E	1	213
140	8C	214	FF	ODD	ADC H	1	214
141	8D	215	CR	EVEN	ADC L	1	215
142	8E	216	SO	EVEN	ADC M	2	216
143	8F	217	SI	ODD	ADC A	1	217
144	90	220	DLE	EVEN	SUB B	1	220
145	91	221	DC1	ODD	SUB C	1	221
146	92	222	DC2	ODD	SUB D	1	222
147	93	223	DC3	EVEN	SUB E	1	223
148	94	224	DC4	ODD	SUB H	1	224
149	95	225	NAK	EVEN	SUB L	1	225
150	96	226	SYN	EVEN	SUB M	2	226
151	97	227	ETB	ODD	SUB A	1	227
152	98	230	CAN	ODD	SBB B	1	230
153	99	231	EM	EVEN	SBB C	1	231
154	9A	232	SUB	EVEN	SBB D	1	232
155	9B	233	ESC	ODD	SBB E	1	233
156	9C	234	FS	EVEN	SBB H	1	234
157	9D	235	GS	ODD	SBB L	1	235
158	9E	236	RS	ODD	SBB M	2	236
159	9F	237	US	EVEN	SBB A	1	237
160	A0	240	SP	EVEN	ANA B	1	240
161	A1	241	!	ODD	ANA C	1	241
162	A2	242	"	ODD	ANA D	1	242
163	A3	243	#	EVEN	ANA E	1	243
164	A4	244	$	ODD	ANA H	1	244
165	A5	245	%	EVEN	ANA L	1	245
166	A6	246	&	EVEN	ANA M	2	246
167	A7	247	'	ODD	ANA A	1	247
168	A8	250	(	ODD	XRA B	1	250
169	A9	251	)	EVEN	XRA C	1	251
170	AA	252	*	EVEN	XRA D	1	252
171	AB	253	+	ODD	XRA E	1	253
172	AC	254	,	EVEN	XRA H	1	254
173	AD	255	-	ODD	XRA L	1	255
174	AE	256	.	ODD	XRA M	2	256
175	AF	257	/	EVEN	XRA A	1	257
176	B0	260	0	ODD	ORA B	1	260
177	B1	261	1	EVEN	ORA C	1	261
178	B2	262	2	EVEN	ORA D	1	262
179	B3	263	3	ODD	ORA E	1	263
180	B4	264	4	EVEN	ORA H	1	264
181	B5	265	5	ODD	ORA L	1	265
182	B6	266	6	ODD	ORA M	2	266
183	B7	267	7	EVEN	ORA A	1	267
184	B8	270	8	EVEN	CMP B	1	270
185	B9	271	9	ODD	CMP C	1	271
186	BA	272	:	ODD	CMP D	1	272
187	BB	273	;	EVEN	CMP E	1	273
188	BC	274	<	ODD	CMP H	1	274
189	BD	275	=	EVEN	CMP L	1	275
190	BE	276	>	EVEN	CMP M	2	276
191	BF	277	?	ODD	CMP A	1	277

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
192	C0	300	@	EVEN	RNZ	1/3	300
193	C1	301	A	ODD	POP B	3	301
194	C2	302	B	ODD	JNZ aa	3	302
195	C3	303	C	EVEN	JMP aa	3	303
196	C4	304	D	ODD	CNZ aa	3/5	304
197	C5	305	E	EVEN	PUSH B	3	305
198	C6	306	F	EVEN	ADI v	2	306
199	C7	307	G	ODD	RST 0	3	307
200	C8	310	H	ODD	RZ	1/3	310
201	C9	311	I	EVEN	RET	3	311
202	CA	312	J	EVEN	JZ aa	3	312
203	CB	313	K	ODD			313
204	CC	314	L	EVEN	CZ aa	3/5	314
205	CD	315	M	ODD	CALL aa	5	315
206	CE	316	N	ODD	ACI v	2	316
207	CF	317	O	EVEN	RST 1	3	317
208	D0	320	P	ODD	RNC	1/3	320
209	D1	321	Q	EVEN	POP D	3	321
210	D2	322	R	EVEN	JNC aa	3	322
211	D3	323	S	ODD	OUT v	3	323
212	D4	324	T	EVEN	CNC aa	3/5	324
213	D5	325	U	ODD	PUSH D	3	325
214	D6	326	V	ODD	SUI v	2	326
215	D7	327	W	EVEN	RST 2	3	327
216	D8	330	X	EVEN	RC	1/3	330
217	D9	331	Y	ODD			331
218	DA	332	Z	ODD	JC aa	3	332
219	DB	333	[	EVEN	IN v	3	333
220	DC	334	\	ODD	CC aa	3/5	334
221	DD	335	]	EVEN			335
222	DE	336	↑	EVEN	SBI v	2	336
223	DF	337	←	ODD	RST 3	3	337
224	E0	340	`	ODD	RPO	1/3	340
225	E1	341	a	EVEN	POP H	3	341
226	E2	342	b	EVEN	JPO aa	3	342
227	E3	343	c	ODD	XTHL	5	343
228	E4	344	d	EVEN	CPO aa	3/5	344
229	E5	345	e	ODD	PUSH H	3	345
230	E6	346	f	ODD	ANI v	2	346
231	E7	347	g	EVEN	RST 4	3	347
232	E8	350	h	EVEN	RPE	1/3	350
233	E9	351	i	ODD	PCHL	1	351
234	EA	352	j	ODD	JPE aa	3	352
235	EB	353	k	EVEN	XCHG	1	353
236	EC	354	l	ODD	CPE aa	3/5	354
237	ED	355	m	EVEN			355
238	EE	356	n	EVEN	XRI v	2	356
239	EF	357	o	ODD	RST 5	3	357
240	F0	360	p	EVEN	RP	1/3	360
241	F1	361	q	ODD	POP PSW	3	361
242	F2	362	r	ODD	JP aa	3	362
243	F3	363	s	EVEN	DI	1	363
244	F4	364	t	ODD	CP aa	3/5	364
245	F5	365	u	EVEN	PUSH PSW	3	365
246	F6	366	v	EVEN	ORI v	2	366
247	F7	367	w	ODD	RST 6	3	367
248	F8	370	x	ODD	RM	1/3	370
249	F9	371	y	EVEN	SPHL	1	371
250	FA	372	z	EVEN	JM aa	3	372
251	FB	373	{	ODD	EI	1	373
252	FC	374	|	EVEN	CM aa	3/5	374
253	FD	375	}	ODD			375
254	FE	376	~	ODD	CPI v	2	376
255	FF	377	DEL	EVEN	RST 7	3	377

TASWORD TWO HELP FILE #1

EDIT - help page
CAPS LOCK - capitals lock
True Video - Cursor to word left
Inv. video - " " " right
ARROWS - cursor movement
GRAPHICS - printer control characters (not used with M100)
DELETE - delete character
<= - move line left
<> - center line
=> - move line right
AND - insert line/character
OR - go to end of text
AT - " " start " "
STOP - save/load/print text
NOT - delete line
STEP - reform to end of para.
TO - schroll down
THEN - scroll up
ENTER - Start of next line
CAPS + symbol shift - enter or leave extended mode

GRAPHICS CODES

ON	EPSON FX80	OFF	
	Enlarged		key 1
	Condensed		key 2
	Underline		key 3
	Italics		key 4
	Emphasised		key 5
	Double Strike		key 6
	Elite		key 7
	Proportional		key 8

Tasword Two Help File #2

Extended Mode keys:

Scrolling
F - fast scroll down
G - fast scroll up

ZX Printer

P - Print text file
L - large printing on marker
K - large printing off marker

Miscellaneous
C - change window on text
X - clear text file
R - replace or find text
I - insert mode on/off
EDIT - help page
Arrows - cursor movement

Formatting:
E - right justify on/off
W - word-wrap on/off
J - justify line
H - UnJustify line

MARGINS
A - set left margin AT cursor
S - Reset margins to normal
D - Set right margin at cursor

BLOCK COMMANDS

B - mark beginning of block
V - mark end of block of text
N - copy marked block to cursor
M - move marked block to cursor

Symbol Shift and key for these characters:

[ ] © ^ | \ { }

U10

1	3.6v
2	1.7v
3	1.5v
4	1.75v
5	1.75v
6	2.45v
7	2.35v
8	0v GND
9	2.15v
10	~~2.35~~ 2.35v
11	1.7
12	2.45
13	2.1v
14	2.35v
15	0
16	4.9v Vcc

(U2) U21 QUAD TWO INPUT NAND GATES

	PIN	VOM READING	LOGIC PROBE READING
SIDE1:	1	4.6V	
	2	3V	
	3	.15V	
	4	1.8V	
	5	4.9V	
	6	2.5V	
	7	ØV (GND)	
SIDE2:	8	3.8V	
	9	2.5V	
	10	.15V	
	11	.15V	
	12	3.8V	
	13	4.9V	
	14	4.9V (Vcc)	

NOTES: VERSION A BOARD
10 Ω RESISTOR IN-LINE WITH +5VDC POWER SUPPLY TO THE SCLA CHIP

IN 73 = 0 with Modem Connected
IN 77 = 0 with Modem Connected
IN 87 = 255 no centronics interface
IN EF = 255 no Micro-drive
IN F4 = 00 Dock port off
IN F5 = 255 Sound chip off
IN F6 = 255 " " "
IN F7 = 255 No Micro-drive
IN FB = 126 Printer off, 58 Printer on
IN FC = 255 No Expansion Banks
IN FD = 255 No Expansion Banks
IN FE = 31 Keyboard
IN FF = 0 Regular Video mode

BAD1: [illegible]
BAD3: [illegible]

BAD1: B2C2
BAD3: B2CA

PASSED

WORKS

SPLIT SUBROUTINE

ENTRY:

B = Byte to disassemble

Exit

128 64 32 16 8 4 2 1
1 1

76 holds bits 7,6
543 holds bits 5,4,3
210 holds bits 2,1,0
54 holds bits 5,4
3 holds bit 3

B is intact!!

~~LD B,A AND A,B~~

```
LD A,B
AND C0
~~LD (76),A  Bits 7 and 6~~
RLCA   ;}shift to bits 1 and 0
RLCA   ;
LD (76),A ; Bits 7 and 6
LD A,B
AND 38
RRCA   ;}
RRCA   ; shift to bits 2,1, and 0
RRCA   ;}
LD (543),A  ;bits 5,4, and 3
LD A,B
AND 07
LD (210),A  ;bits 2,1, and 0
```

```
7  6  5  4  3  2  1  0
128 64 (36 16) 8  :  :  :
```

```
LD A,B
AND 30
RRCA
RRCA  ; } shift to bits 1 and 0
RRCA  ; }
RRCA
LD (54),A   ; bits 5 and 4
LD A,B
AND 08
RRCA
RRCA  ; } shift to bit 0
RRCA
LD (3),A   ; bit 3
RET   ; finished
```

```
128 64 32 16 8 4 2 1
 0  0  1  1  0 0 0 0   = 0
                      = 6
                      = 0
                      = 3
                      = 0
```

```
60928   76
    9   543
   30   210
   31    54
   32     3
```

```
        LD A,B
        CP 76
        JP NZ, NOT HALT (06F2)
        LD A, 'H' (48h)
        LD HL, DISSTRING (0674)
        LD (HL), A
        LD A, 'A' (41h)
        INC HL
        LD (HL), A
        LD A, 'L' (4Ch)
        INC HL
        LD (HL), A
        LD A, 'inverse T' (04)
        INC HL
        LD (HL), A
        JP PRINT (    )
NOT     LD HL, DATA (address) (    )
HALT    PUSH HL
(06F2)  LD A, B
(55 026)CP 08
        RET Z
        CP 10
        RET Z
        CP 18
        RET Z
        CP 28
        RET Z
        CP 38
        RET Z
        CP CB
        RET Z
        CP D9
        RET Z
        CP ED
        RET Z
        CP FD
        RET Z
```

```
                POP HL
                LD A,(76) variable (D6CE)
                CP ØØ
                JP NZ,(76)<>Ø
                LD A,(21Ø) variable (D6DØ)
                CP ØØ
                JP NZ,(21Ø)<>Ø (D75B)
                LD A,(543) variable (D6CF)
                CP 2Ø
                JP NZ,(543)<>2Ø (     )
                LD A,'R' (52h)
                LD HL, Dis string (D674)
                LD (HL),A
'IM'            LD A,'I' (49h)
(D731)          INC HL
(55Ø89)         LD (HL),A
                LD A, 'inverse M' (CDh)
                INC HL
                LD (HL),A
                JP PRINT (     )
(543)<>2Ø       CP 3Ø
(D73C)          JP NZ, NOP
(551ØØ)         LD A,'S' (53h)
                LD HL, DIS STRING (D674)
                LD (HL),A
                JP 'IM'          (D731)
NOP             LD A,'N' (4Eh)
(D74A)          LD HL, DIS STRING
(55114)         LD (HL),A
                LD A, 'O' (4Fh)
                INC HL
                LD (HL),A
                LD A, 'inverse P' (CDØh)
                INC HL
                LD (HL),A
                JP PRINT (     )
```

```
(210)<>Ø    CP Ø1
(D75B)      JP NZ,(210)<>1   (    )
(55131)     LD A,(3) variable
            CP ØØ
            JP NZ,(3)=1      (    )
            LD A,'L' (4Ch)
            LD HL,DIS STRING (D674)
            LD (HL),A
            LD A,'D' (44h)
            INC HL
            LD (HL),A
SPACER      LD A,'space' (2Øh)
(D772)      INC HL
(55154)     LD (HL),A
(D777) (55159) INC HL
BC/DE/HL/SP LD A,(54) variable (D6D1)
  BC/DE/    CP ØØ
  HL/SP1
  (D77A)    JP NZ,Not BC     (    )
  (55162)   LD A,'B' (42h)
            LD (HL),A
            LD A,'C' (43h)
            INC HL
            LD (HL),A
ADDRESS PRINT LD A,',' comma (2Ch)
(D786)      INC HL
(55174)     LD (HL),A
            LD A,(bit)  (D6D3)
            CP Ø1
            JP Z,EXTRA  (    )
            INC HL
            LD A,Ø2
            LD (bit),A  (D6D3)
            JP PRINT    (    )
EXTRA       XOR A
            LD (bit),A
            LD A,AØ (inverse space)
            LD (HL),A
```

```
        JP PRINT
NOT BC  CP 01
        JP NZ, NOT DE
        LD A, 'D'
        LD (HL), A
        LD A, 'E'
        INC HL
        LD (HL), A
        JP ADDRESS PRINT
NOT DE  CP 02
        JP NZ, 'SP'
        LD A, 'H'
        LD (HL), A
        LD A, 'L'
        INC HL
        LD (HL), A
        JP ADDRESS PRINT
'SP'    LD A, 'S'
        LD (HL), A
        LD A, 'P'
        INC HL
        LD (HL), A
        JP ADDRESS PRINT
(3)=1   LD HL, DES STRING
        LD A, 'A'
        LD (HL), A
        LD A, '0'
        INC HL
        LD (HL), A
        INC HL
        LD (HL), A
        LD A, 'space'
        INC HL
        LD (HL), A
        LD A, 'H'
        INC HL
        LD (HL), A
```

```
                LD A, 'L'
                INC HL
                LD (HL), A
                LD A, ',' comma
                INC HL
                LD (HL), A
                INC HL
                LD A, 01
                LD (bit), A
                JP BC/DE/HL/SP
(210)<>1        CP 02
                JP NZ, (210)<>2
                LD A, 'L'
                LD HL, DIS STRING
                LD (HL), A
                LD A, 'D'
                INC HL
                LD (HL), A
                LD A, 'space'
                INC HL
                LD (HL), A
                INC HL
SEARCH          LD A, (543) variable
                RRCA
                RRCA
                RRCA
                JR C, LOOPER 2
                RLCA
                RLCA
                RLCA
                PUSH DE
                LD DE, DATA STRING 1
SEARCH1         CP 00
                JP NZ, LOOPER 1
LOOPER          XOR A
                LD A, (DE)
                RLCA
```

```
                JP C, TERMINATOR1
                RRCA
                LD (HL), A
                INC HL
                JP LOOPER
TERMINATOR1     RRCA
                LD (HL), A
                POP DE
                JP PRINT
LOOPER1         PUSH AF
LOOP            XOR A
                LD A, (DE)
                RLCA
                JP C, INCER
                INC DE
                JP LOOP
INCER           POP AF
                INC DE
                DEC A
                PUSH AF
                CP ΦΦ
                JP NZ, LOOP
                POP AF
                JP LOOPER
DATA STRING1    DEFS '(BC), [A] A, (BC[J] CDE), [A] A, CDE[J]'  ; [] = TERMINATOR BYTE
LOOPER2         POP DE
                ~~CP Φ4~~
                ~~JP NZ, NOT 4~~
                ~~LD (bit), A~~        (210)<>2
                ~~JP PRINT~~
                ~~CP Φ5~~
                ~~JP NZ, NOT 5~~
                ~~JP BYPASS~~
                ~~CP Φ6~~              ENT1
                ~~JP~~ .
                LD (bit), A
                JP Print
                CP Φ3
                JP NZ, (210) <> 3
                LD A, (3) variable
                CP ΦΦ
                JP NZ, (3) = 1A
                LD A, 'I'
                LD AL, DIS STRING
                LD (HL), A
                LD A, 'N'
```

```
        INC HL
        LD (HL),A
SHORT   LD A,'C'
        INC HL
        LD (HL),A
        LD A,'space'
        INC HL
        LD (HL),A
        INC HL
        LD A,(mode)
        CP 00
        JP NZ,mode2
MODE1   LD A,01
        LD (bit),A
        JP BC/DE/HL/SP
(3)=1A  LD A,'D'
        LD HL,DES STRING
        LD (HL),A
        LD A,'E'
        INC HL
        LD (HL),A
        LD A,'C'
        INC HL
        LD (HL),A
        LD A,'space'
        INC HL
        LD (HL),A
        INC HL
        LD A,(mode)
        CP 00
        JP NZ,mode2
Mode 3  LD A,01
        LD (bit),A
        LD A,(210)
        AND 03
        JP BE/DE/HL/SP1
(210)<>3 CP 04
```

```
        JP NZ, (210) <>4
        LD A, Ø1
        LD (mode), A
        JP Ent1
(210) <> 4  CP Ø5
        JP NZ, (210) <>5
        LD A, Ø1
        LD (mode), A
        JP (3) = 1A
(210) <> 5  CP Ø6
        JP NZ, =7
        LD A, 'L'
        LD HL, DIS STRING
        LD (HL), A
        LD A, 'D'
        INC HL
        LD (HL), A
        LD A, 'space'
        INC HL
        LD (HL), A
        INC HL
B/C/D/E     LD A, (543)
B/C/D/E1    CP ØØ
        JP NZ, NOT B
        LD A, 'B'
        LD (1+L), A
Add Print   LD A, ',' comma
        INC. HL
        LD (1+L), A
        LD A, (bit)
        CP Ø1
        JP Z, No Add
        INC HL
        LD A, Ø1
        LD (bit), A
        JP Print
No Add      XOR A
```

```
        LD (bit), A
        LD A, 'inverse space'
        LD (HL), A
        JP Print
NOT B   CP 01
        JP NZ, NOT C
        LD A, 'C'
        LD (HL), A
        JP Add Print
NOT C   CP 02
        JP NZ, NOT D
        LD A, 'D'
        LD (HL), A
        JP Add Print
NOT D   CP 03
        JP NZ, NOT E
        LD A, 'E'
        LD (HL), A
        JP Add Print
NOT E   CP 04
        JP NZ, NOT H
        LD A, 'H'
        LD (HL), A
        JP Add Print
NOT H   CP 05
        JP NZ, NOT L
        LD A, 'L'
        LD (HL), A
        JP Add Print
NOT L   CP 06
        JP NZ, NOT (HL)
        LD A, 'C'
        LD (HL), A
        LD A, 'H'
        INC HL
        LD (HL), A
        LD A, 'L'
```

```
        INC HL
        LD (HL), A
        LD A, ')'
        INC HL
        LD (HL), A
        JP Add Print
Not (HL) LD A, 'A'
        LD (HL), A
        JP Add Print
Mode2   LD A, 01
        LD (bit), A
        JP BIC/DE
=7      Push DE
        LD DE, DATA STRING2
        LD A, (543)
        JP Search 1
DATA STRING2 DEFS 'RLC(A)RRC(A)RL(A)RR(A)SLA(A)CPL(E)SCF(F)CCF' □=Terminator Byte
```

NAME	EQU
DES STRING	D6C2
(76)	D6D1
(543)	D6D2
(210)	D6D3
(54)	D6D4
(3)	D6D5
(bit)	D6D6
(mode)	D6D7
NOT HALT	D6F2
'IM'	D731
(543) <> 2Ø	D73C
NOP	D74A
(210) <> Ø	D75B
SPACER	D772
BC/DE/HL/SP	D777
BC/DE/HL/SP1	D77A
Address Print	D786
EXTRA	D79B
NOT BC	D7A5
NOT DE	D7B4
ISP'	D7C3
(3) =1	D7CD
(210) <> 1	D7F2
SEARCH	D8Ø6
SEARCH 1	D816
LOOPER	D81B
TERMINATOR1	D827
LOOPER 1	D82D
LOOP	D82E
INCER	D838
DATA STRING 1	D845
(210) <> 2	D864
ENT 1	D871
SHORT	D87B
MODE 1	D88C
(3) = 1A	D894

NAME	EQU
MODE 3	D8AF
(210) <> 3	D8BC
(210) <> 4	D8C9
(210) <> 5	D8D6
B/C/D/E	D8EA
B/C/D/E1	D8ED
Add Print	D8F5
No ADD	D9ØA
NOT B	D914
NOT C	D91F
NOT D	D92A
NOT E	D935
NOT H	D94Ø
NOT L	D94B
NOT (HL)	D962
MODE 2	D968
=7	D97Ø
DATA STRING 2	[illegible] D97D
LOOPER 2	D85D
SPECIAL	D997

41922

hi 163 / Low 194

hi 196 / Lo 110

ERROR: 0932 must JP 08F5

NOT 09F5

ERROR2: 080C - Retrpt to 0997 turn 18' to a 19'

ERROR3: NOT D has a bad Jump!!!

1) If byte is 76 then instruction is 'HALT'
2) If byte is Ø8, 1Ø, 18, 28, 38, CB, D9, ED, FD, instruction is 'DATA'
3) If (76) = zero then...
   If (21Ø) = zero then...
      If (543) = 2Øh, the instruction is 'RIM'
      If (543) = 3Øh, the instruction is 'SIM'
      If (543) = ØØh, the instruction is 'NOP'
        cannot be anything else...
   If (210) = one then...
      If (3) = zero, the instruction is LD s(54), V^{LOW}, V^{HIGH}
      If (3) = one, the instruction is ADD HL, s(54)
   If (210) = two then...
      the instruction is LD with the (543) item in list
        (BC), A/A, (BC) / (DE), A/A, (DE) / (VV), HL/HL, (VV) /
        (VV); A/A, (VV)
   If (210) = three then...
      If (3) = zero, the instruction is INC s(54)
      If (3) = one, the instriction is DEC s(210)

Bit = Ø for 1 byte, 1 = 2 bytes, 2 = 3 bytes

```
>4 = (VV), HL
5 = HL, (VV)
6 = (VV), A
7 = A, (VV)

JP NZ, NOT HALT      : not halt
LD A, 'H'            : must be halt
LD HL, Dis string    ; besinning of Dis string
LD (HL), A
LD A, 'A'
INC HL
LD (HL), A
LD A, L'
INC HL
```

HALT

```
LD (HL), A
LD A, 'inverst T' ; terminator
INC HL
LD (HL), A
JP PRINT - Print it
```

(bit = Ø one byte)

```
LD A, B
CP 76
~~JP Z, HALT~~
NOT HALT > LD HL, DATA (address)
PUSH HL
LD A, B
CP Ø8 2
RET Z 1 (indirect jump to DATA)
CP 1Ø
RET Z
CP 18
RET Z
CP 28
RET Z
CP 38
RET Z
CP CB
RET Z
CP D9
RET Z
CP ED
RET Z
CP FD
RET Z
POP HL          ; kill DATA address
LD A, (76)
CP ØØ
JP NZ, F<>Ø     ; F is not Ø
LD A, (21Ø)
CP ØØ
JP NZ, H<>Ø     ; H is not Ø ;
LD A, (543)     ; H is zero !
CP 20 G
JP NZ, G<>32    ; G is not 2Øh
LD A, 'R'       ; G is 20h (must be RIM)
LD HL, DIS string ØØ (begining of Dis string)
LD (HL), A      ; put 'R' there
```

ALL 'DATA'

```
'IM'    LD A,'I'
        INC HL
        LD (HL),A
        LD A,'inverse M'  (bit 7 set as terminator)
        INC HL
        LD (HL),A
G       JP Print -> Print MNMENOIC  ↙ one byte (bit = Ø)
H<>32   CP 30  ; is G 30 h?
        JP NZ, G<>48   ; No G is no 48d
        LD A,'S'       ; Yes, must be (SIM)
        LD HL, Dis string  ; beginnig of Dis string
        LD (HL),A      ; Pus 'S' there
G       JP IM          ; And finish through 'RIM'
H<>48   LD A,'N'       ; Must be 'NOP'
        LD HL, Dis string  ; Beginning of Dis string
        LD (HL),A      ; Put 'N' there
        LD A,'O'
        INC HL
        LD (HL),A
        LD A,'P' terminator inverse   } Put 'O' and 'P'
        INC HL
        LD (HL),A
        JP Print -> Print it  ↙ one byte (bit = Ø)
H<>Ø    CP Ø1          ; is H 1?
        JP NZ, H<>1    ; No pass -> -> ->
        LD A,(3)       ; Yes H=1!!
        CP ØØ          ; is (3) = Ø?
        JP NZ, (3)=1   ; No, pass to, (3)=1
        LD A,'L'       ; Yes (3)=Ø!!, must be 'LD'
        LD HL, Dis string
        LD (HL),A
        LD A,'D'
        INC HL
        LD (HL),A
Spacer: LD A,'space'
        INC HL
        LD (HL),A
```

Put 'LD_' there

PAGE 2

```
                INC HL
BC/DE/HL/SP     LD A,(34)       ;check which Reg is loading
BC/DE/HL/SP1    CP 00           ;is it 'BC'?
                JP NZ,NOT BC    ;No PASS →
                LD A,'B'        ;Yes 'BC' it is!
                LD (HL),A
                LD A,'C'
                INC HL          } PUT 'BC' there!
Address Print   LD (HL),A
                LD A,',' comma  } Put a ',' there!!
                INC HL
                LD (HL),A <<<<<<<
                INC HL
                LD A,02
                LD (bit),A      ;set flag for 3 byte instruction
                JP Print        ;Print full MNEMONIC with address
NOT BC          CP 01           ;Is it 'DE'
                JP NZ,NOT DE    ;No Pass →
                LD A,'D'        ;Yes, 'DE' it is!
                LD (HL),A
                LD A,'E'        } Put 'DE' there
                INC HL
                LD (HL),A
                JP Address Print ;Jump through 'BC' to finish
NOT DE          CP 02           ;IS 'HL'
                JP NZ,'SP'      ;No - must be 'SP'
                LD A,'H'        ;Yes - 'HL' it is!!
                LD (HL),A
                LD A,'L'        } Put 'HL' there
                INC HL
                LD (HL),A
                JP Address Print ;Jup through 'BC' to finish
'SP'            LD A,'S'        ;must be 'SP'
                LD (HL),A
                LD A,'P'        } Pus 'SP' there!
                INC HL
                LD (HL),A
```

LD A,(bit)
CP 01
JP Z, EXTRA
JP PRINT

EXTRA: XOR A
LD (bit),A
JP PRINT

LD A,A0
LD (HL),A

```
                Jump Address Print      ; and finish through 'BC' routine
(3) = 1         LD HL, Dis String       ; must be 'ADD'
                LD A, 'A'
                LD (HL), A
                LD A, 'D'
                INC HL
                LD (HL), A
                INC HL
                LD (HL), A
                LD A, 'space'
                INC HL
                LD (HL), A
                LD A, 'H'
                INC HL
                LD (HL), A
                LD A, 'L'
                INC HL
                LD (HL), A
                LD A, ',' comma
                INC HL
                LD (HL), A
                INC HL
                LD A, Φ1
                LD (IX), A
                JP BC/DE/HL/SP
(210) <> 1      CP Φ2
                JP NZ, (210) <> 2
                LD A, 'L'
                LD HL, Dis string
                LD (HL), A
                LD A, 'D'
                INC HL
                LD (HL), A
                LD A, 'space'
                INC HL
                LD (HL), A
                INC HL
```

Put 'ADD ' there!!'

Put '_HL, ' there

We only want the Reg and not the address!!!

;-find and Print reg/MNMONEC
-;S (210) equal to 2?
-No Pass →
-Yes must be LD xx, xx

Put 'LD_' there

3 4 5

RRCA
RRCA } see if greater than 3
RRCA
JP C, Looper 2 — Yes - jump to looper 2
RLCA
RLCA } NO — replace bits
RLCA

PAGE 3

~~Push HL~~ ~~Save address~~

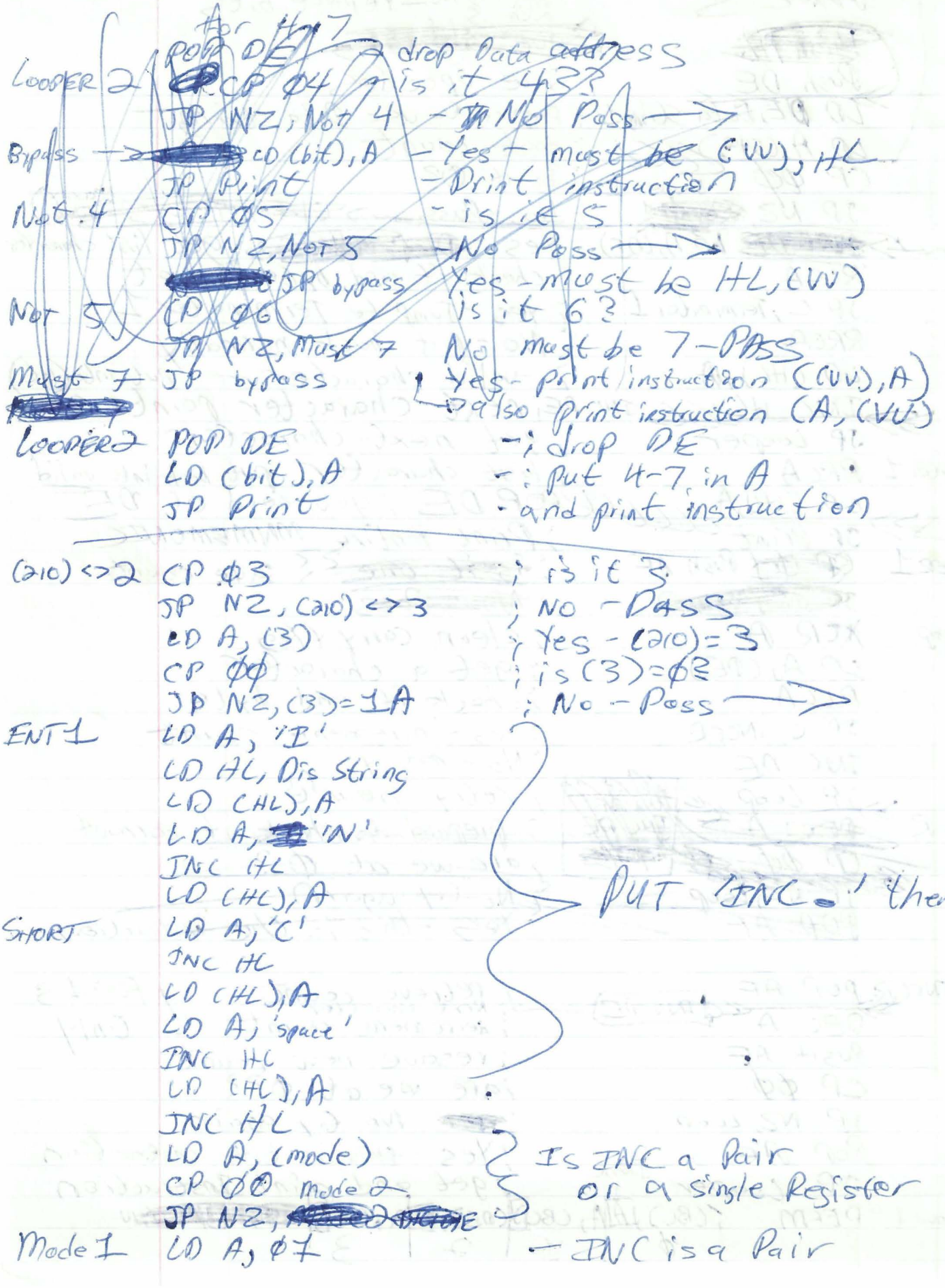
Push DE ; Save opcodes
LD DE, Data String 1 ; prepare to use Data string —
LD A, (5H3) ; get count
CP ØØ Looper 1 ; is it Ø?
JP NZ, ~~Looper 1~~ ; No - Pass → ~~Clean Flag~~ (carry flag)
Looper → ~~[illegible]~~ LD A, (DE) ; Yes - ~~[illegible]~~ get first character
RLCA ; check if end byte is set
JP C, Terminator 1 ; yes - Jump to Terminator 1
RRCA ; No - put back normally
LD (HL), A ; valid character — put into (HL)
~~INC HL~~ INC DE ; next character pointer
JP Looper ; get next character
Terminator 1 RRCA ; last character - put back into valid
LD (HL), A POP DE ; get reid of DE
JP Print ; Print Entire MNMEMONIC
LOOPER 1 ~~CP Ø1~~ Push AF ; ~~is it one~~ ?? save count
~~JP NZ, Looper 2~~ ; ~~No - Pass~~
Loop XOR A ; clean carry flag
LD A, (DE) ; get a character
RLCA ; check if end byte
JP C, INCER ; yes - increment count
INC DE ; No - retry
JP Loop ; retry now ?
INCER: ~~DEC A~~ ~~POP AF~~ ~~PUSH AF~~ ~~INC DE~~ ~~; prepare to check if correct~~
~~CP ØØ~~ ~~; are we at Ø?~~
~~JP NZ, Loop~~ ~~; No try again~~
~~POP AF~~ ~~; Yes - this is the instruction~~

INCER: POP AF ; retreve count
INC DE → ; Next character
DEC A ; decrement count
PUSH AF ; resave new count
CP ØØ ; are we at Ø??
JP NZ, Loop ; No try again
POP AF ; Yes - this is the instruction
JP Looper ; get and print Instruction

FOR 1-3 Only

Inverse

DATA STRING 1 DEFM (BC), A, (BC), (DE), A, (DE) ~~[illegible]~~
Ø 1 2 3


Looper 2
POP DE
CP 04
JP NZ, Not 4
Bypass
CD (bit), A
JP Print
Not 4
CP 05
JP NZ, Not 5
JP bypass
Not 5
CP 06
JP NZ, Must 7
Must 7
JP bypass
Looper2
POP DE
LD (bit), A
JP Print
drop Data address
is it 4?
- No Pass
- Yes - must be (VV), HL
- Print instruction
- is it 5
- No Pass
Yes - must be HL, (VV)
is it 6?
No must be 7 - PASS
Yes - print instruction (VV), A
also print instruction (A, (VV)
- drop DE
- Put 4-7 in A
- and print instruction
(210) <>2
CP 03
JP NZ, (210) <>3
LD A, (3)
CP 00
JP NZ, (3)=1A
; is it 3
; No - PASS
; Yes - (210)=3
; is (3)=02
; No - Pass
ENT 1
LD A, 'I'
LD HL, Dis String
LD (HL), A
LD A, 'N'
INC HL
LD (HL), A
SHORT
LD A, 'C'
INC HL
LD (HL), A
LD A, 'space'
INC HL
LD (HL), A
INC HL
LD A, (mode)
CP 00 mode 2
JP NZ,
Mode 1
LD A, 07
PUT 'INC.' there
Is INC a Pair
or a single Register
- INC is a Pair

PAGE 4

```
        LD (bit),A      ;- Selection bit
        JP BC/DE/HL/SP  - Print instruction
```

```
(3)=1A  LD A,'D'        ;(3) must be one!
        LD HL,Dis string
        LD (HL),A
        LD A,'E'
        INC HL
        LD (HL),A
        LD A,'C'
        INC HL
        LD (HL),A       PUT "DEC_" there
        LD A,'space'
        INC HL
        LD (HL),A
        INC HL
        LD A,(mode)     Is DEC a pair
        CP 00           Mode 2  or a single Register
        JP NZ,
Mode 3  LD A,01         DEC is a PAIR
        LD (bit),A
                        LD A,(210)
        AND 03          - mask out unneeded bits
        JP BC/DE/HL/SP1 - Print instruction and Register
(210)<>3 CP 04          ;is it 4
        JP NZ,(210)<>4  ;No Pass
                        ;Yes (210)=4
        LD A,01         ; this is a single register
        LD (mode),A     ;goto INC ->  Print Instruction and Register
        JP ENT1
(210)<>4 CP 05          ;is it 5?
        JP NZ,(210)<>5  ;No Pass ->
        LD A,01         ;Yes (210)=5
        LD (mode),A     ;this is a single register
        JP (3)=1A       ;goto DEC ->  Print Instruction and Register
(210)<>5 CP 06          ;is it 6?
        JP NZ,7         ;no - Pass ->
'LD-'   LD A,'L'        ;Yes - must be 6
```

```
        LD HL, DIS STRING
        LD (HL), A
        LD A, '0'
        INC HL
        LD (HL), A
        LD A, 'space'
        INC HL
        LD (HL), A
        INC HL
BICIDIE LD A, (543)      ; check which Reg is loading
BICIDIE1 CP 00           ; is it B?
        JP NZ, Not B     ; No Pass ->
        LD A, 'B'        ; Yes
        LD (HL), A       ; Put a 'B' there!
Add Print LD A, ';' comma ; Put a ';' there!
        INC HL
        LD (HL), A
        LD A, (bit)      ; is no address desired?
        CP 01            ; Yes - Jump
        JP Z, No Add     ; No set for single address
        LD A, 01         ; - 2 byte instruction
        LD (bit), A      ; and Print instruction/address
        JP Print
No Add  XOR A            ; No address is desired
        LD (bit), A      ; one byte instruction
        LD A, 'inverse space'
        LD (HL), A       ; Jump and Print
        JP Print
Not B   CP 01            ; is it 'C'?
        JP NZ, Not C     ; No - Pass ->
        LD A, 'C'        ; Yes - 'C' it is
        LD (HL), A       ; Put a 'C' there,
        JP Add Print     ; and Finish
Not C   CP 02            ; is it 'D'?
        JP NZ, Not D     ; No Pass ->
        LD A, 'D'        ; Yes
        LD (HL), A       ; Put a 'D' there!
```

CP 02
RET Z

INC HL

```
        JP Add Print      - and finish instruction
NOT D   CP 03             ;is it 'E'
        JP NZ, Not E      ;No Pass -->
        LD A, 'E'         ;Yes Put an 'E' there
        LD (HL), A        ;
        JP Add Print      ;and finish -->
NOT E   CP 04             ;is it 'H'
        JP NZ, Not H      ;No Pass -->
        LD A, 'H'         ;Yes Put an 'H' there
        LD (HL), A        ;
        JP Add Prit       ;and Finish -->
NOT H   CP 05             ;is it 'L'
        JP NZ, Not L      ;No - Pass -->
        LD A, 'L'         ;Yes Put an 'L' there
        LD (HL), A        ;
        JP Add Print      ;and Finish
NOT L   CP 06             ;is it '(HL)'?
        JP NZ, Not (HL)   ;No - Pass -->
        LD A, '('         ;
        LD (HL), A        ;
        LD A;'H'          ;
        INC HL            ;
        LD (HL), A        ;
        LD A, 'L'         ;
        INC HL            ;
        LD (HL), A        ;
        LD A, ')'         ;
        INC HL            ;
        LD (HL), A        ;
        JP Add Print      ;and Finish -->
NOT(HL) LD A, 'A'         ;must be '7'
        LD (HL), A        ;Put a 'A' there
        JP Add Print      ;and Finish... -->
Mode2   LD A, 01          ;we only want instruction
        LD (bit), A       ;no Addresses
        JP B/C/D/E        ;--> Do it (Finish... -->)
```

Put '(HL)' there!

=7 PUSH DE ; save it ; must be 7
LD DE, Data string 2
LD A, (543) ; get count → Set up Dis string
JP search 1 ; and finish →

DATA STRING 2 DEFS 'RLCA RRCA RLA RRA DAA CPL SCF CCF'
(76) <> 0

SPECIAL RLCA
RLCA
RLCA
} Restore correct number
LD (bit), A - but in to (bit)
JP PRINT - and print special instruction

PAGE 6

```
                              → LD A,(mode)
[illegible]   CP ∅1                ; is (76) = 1?
PASB          JP NZ,(76)<>1        ; No - PASS
              LD A,∅2              ; Yes - set up a return
              LD (bit),A           ; to the address in HL
              LD HL,[illegible]    ; by pushing HL on stack
              PUSH HL              ; and RETing
Sho2          JP 'LD'              ; Print 'LD' and first register
[illegible]   INC HL               ; next character space
[illegible]   LD A,∅1              ; we only want
Sho4          LD (bit),A           ; instruction
              LD A,(210)           ; get resister
              JP B/C/D/E1          ; and Finish →
[illegible]   CP ∅2                ; is (76) = 2?
PASC          JP NZ,(76)<>2        ; No - Pass
              LD A,(543)           ; Yes - which instruction?
              CP ∅∅                ; is it ADD A, ?
              JP NZ, NOT ADD A,    ; No PASS →
              LD A,'A'             ; Yes
              LD HL, DIS STRING
              LD (HL),A
              LD A,'D'
              INC HL
              LD (HL),A            Print 'ADD A,
              INC HL
              LD (HL),A
Sho3          LD A,'space'
              INC HL
              LD (HL),A
              LD A,',' comma
              INC HL
              LD (HL),A
              [illegible]          ; next character
              JP [illegible]       ; Finish →
[illegible],  CP ∅1                ; is it ADC A, ?
NADD          JP NZ, NOT ADC       ; No - PASS →
              LD A,'A'             ; Yes
              LD HL, DIS STRING
```

Box (upper insert):

```
[illegible]
CP ∅1
JP NZ,Sho4
[illegible]
LD A,∅1
LD (bit),A
JP PRINT
```

Box (middle insert):

```
LD A,'A'
INC HL
LD (HL),A
```

Box (lower insert):

```
~~LD A,(bit)~~
~~CP ∅1~~
~~JP NZ,Sho2~~
[illegible]
~~JP PRINT~~
JP Sho2
```

```
                LD (HL),A
                LD A,'D'
                INC HL
                LD (HL),A
                ;DEC A
                INC HL
                LD (HL),A
                JP Shor3
                CP 02
NADC            JP NZ,Not Sub
                LD A,'S'
                LD HL, Dis String
                LD (HL),A
                LD A,'U'
                INC HL
                LD (HL),A
                LD A,'B'
                INC HL
                LD (HL),A
Sho5            LD A,'space'
                INC HL
                LD (HL),A
                JP Shor2
NSUB            CP 03
                JP NZ, Not SBC
                LD A,'S'
                LD HL, Dis String
                LD (HL),A
```

Print 'ADC_A,'

-Finish throws Shor3
;is instruction SUB
; No - Pass →

Print 'SUB_'

and

; Finish →
; is instruction SBC_A,
; No - PASS →

```
        LD A,'B'
        INC HL
        LD (HL),A
        [illegible]
        INC A
        INC HL
        LD (HL),A
        JP Shor3
~~Not SBC~~ CP Ø4
NSBC    JP NZ, NOT AND
        LD A,'A'
        LD HL, DIS STRING
        LD (HL),A
        LD A,'N'
        INC HL
        LD (HL),A
        LD A,'D'
        [illegible]
        INC HL
        LD (HL),A
        ~~LD A,'space'~~
        ~~INC HL~~
        ~~LD (HL),A~~
        [illegible]
        ~~JP Shor3~~
~~Not AND~~ CP Ø5
NAND    JP NZ, NOT XOR
        LD A,'X'
        LD HL, DIS String
        LD (HL),A
~~Shor6~~ CP A,'O'
Sho6    INC HL
        LD (HL),A
        LD A,'R'
        INC HL
        LD (HL),A
        ~~LD A,'space'~~ JP shor5 ;Finish →
```

Print 'SBC_A,

;and Finish →
;is it AND?
;No Pass →
;Yes
;

Print 'AND_'

→ JP Shor5

;and Finish
;is it Xor?
;No Pass →

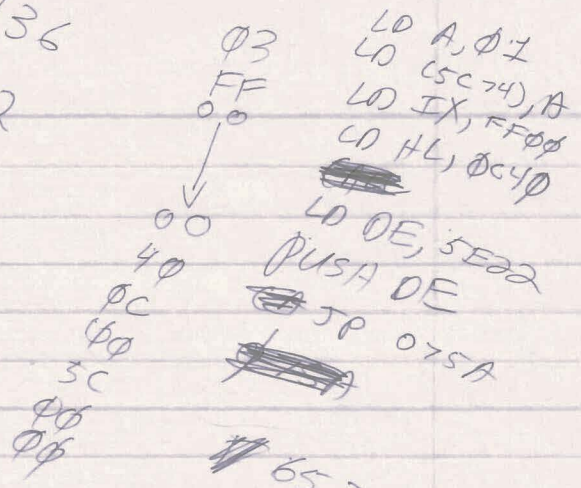
Print 'XOR_'

```
NXOR    CP  06          ; is it OR
        JP  NZ, CP      ; No - Goto 'CP'
        LD  AL, DES string  } Print 'OR_'
        DEC HL
        JP  Short6      } ; Finish ->
'CP'    LD  A, 'C'
        LD  HL, DES string
        LD  (HL), A     } Print 'CP_'
        LD  A, 'P'
        INC HL
        JP  Short5      ; Finish ->
```

PASD

3136
-23552

dele	28ED
sti1	28F8
stik	2902
sti2	2926
ts12	2928
free	2934



Timex 2068 Addss

IN SKIP-OVER subroutine

0083	C0 0C	; Return if DELETE
0085	RET Z	; has been reached (do not print 'DELETE')

IN Token TABLE

0208 → >0226 New Tokens (Delete, ON ERR, STICK, SOUND, FREE, and RESET)

IN Extended mode with shift

0268 → >0281 New TS 2068 codes for above Tokens.
No bytes added, however, ▯ character and the ▯ character no longer available!!

In the keyboard Routines:

Init Subroutine -> Page 1
Timex added:

```
ØD88  LD HL,FUNC (62ØØ)   ;Machine stack at 61FF down
ØD8B  LD (mstb),HL (5CCØ) ;Put 62ØØ (bottom of machine stak) in MSTBOT
ØD8E  DEC HL              ;Point to first byte in machine stack

ØD9A  NOP                 ;Wait 4T states (Effect IM 1)

ØD9F  LD HL,6840          ;New base address for channel information area

ØDD1  XOR A               ;clear A Reg
ØDD2  OUT (FF),A          ;select Normal Video/Keyboard Int/EXROM off
      ~~SET 1 (flgs)(5C3B)~~  ~~;set printer flg to '1' (print to printer)~~
      ~~(CALL clpb (ØA35))~~  ~~;clear printer buffer~~
ØDE3  SET 4,(flgs) (5C3B) ;???

ØDF1  LD HL,ØE ØB         ;source address ØE ØB
ØDF4  LD DE,6ØØØ          ;destination address 6ØØØ
ØDF7  LD BC,ØØ1D          ;length (29 decimal)
ØDFA  LDIR                ;copy it now
ØDFC  CALL 6ØØØ           ;and call this copied Routine
                          Note: Interrupt are still disabled!!!
6ØØØ  LD A,Ø1             ;Load EXROM chunk value
6ØØ2  OUT (F4),A          ;Effect change
6ØØ4  IN A,(FF)           ;Get EXROM bit
6ØØ6  SET 7,A             ;turn on EXROM
6ØØ8  OUT (FF),A          ;Effect change (EXROM ON)
6ØØA  LD HL,1ØØØ          ;source address (in EXROM) 1ØØØ
6ØØD  LD DE,FUNC(62ØØ)    ;destination address 62ØØ
6Ø1Ø  LD BC,Ø63Ø          ;length of Ø63Ø (1584 bytes in decimal)
6Ø13  LDIR                ;block load RAM-RES code into RAM
6Ø15  RES 7,A             ;Turn off EXROM
6Ø17  OUT (FF),A          ;Effect change
6Ø19  XOR A               ;Clear A Reg
6Ø1A  OUT (F4),A          ;Effect chunk select change
6Ø1C  RET                 ;go back to ROM (or called routine)
```

```
ØDFF   LD HL,RETA (65CE)   ; This is the Bank Switch Stack Pointer
ØE02   LD (RETA),HL        ; Put this address in the bottom of Bank Switch stack
ØE05   LD HL,Ø8E7          ; LD HL with EXROM address Ø8E7
ØE08   CALL GOEX (6815)    ; and Goto EXROM with no return
ØE0B
thru   } is a copy of 6ØØØ to 6Ø1C
ØE27
```

In EXROM now - system boot continues
Interrupts still disabled - Bit 4 of (flgs) set still

```
Ø8E7   LD HL,SYSCON (5EEA)   ; HL holds base address of System Config. Table
Ø8EA   LD (sysx),HL (5CBC)   ; and initialize SYSCON varible to it
Ø8ED   CALL bsct (Ø9F4)      ; now call this routine
```

See page 2/3

```
Ø9F4   LD HL,(sysc) (5CBC)   ; HL holds base address for SYS_TAB (Sys. con. table)
Ø9F7   XOR A                 ; Clear A and flags
Ø9F8   LD (maxb),A (5CBE)    ; Max banks initalized to Ø
Ø9FB   LD (63I5),A           ; Bank Switch Maxbank is a copy of Maxbank
Ø9FE   LD DE,erro (ØØØ8)    ; DE holds RST Ø8H (error call)
ØA01   ADD HL,DE             ; add 8 bytes to HL (skip over AROS bytes)
Ø
ØA02   LD E,FF  (dest-bank)  ; LD DE with ØØFF
ØA04   LD D,Ø   (source-bank); one byte at a time
ØA06   PUSH DE               ; save DE
ØA07   LD DE,ØØØ1 (source-addr) ; DE holds 1
ØA0A   PUSH DE               ; SAVE DE
ØA0B   PUSH HL  (dest-addr)  ; SAVE HL (5EF2 is in HL)
ØA0C   LD DE,ØØØ4 (length)   ; DE holds 4
ØA0F   PUSH DE               ; Save DE
ØA10   LD DE,ØØØ1 (direction); DE holds 1 (address from which data is taken)
ØA13   PUSH DE               ; SAVE DE
ØA14   CALL XFER (6722)      ; Transfer these bytes (Get LROS header bytes)
```

See page 4/5

PAGE 9

Init Cont

```
LD A,(HL)              ;get LROS present? byte
CP 01                  ;is LROS present?
JR Z,0A3E (lros)       ;yes jump to 0A3E (lros)
LD (HL),00             ;NO - put 00 in LROS byte
LD E,FF                ;dest-bank is home bank (FF)
LD D,00                ;source-bank is cartridge bank (C0)
PUSH DE                ;save it for XFER_Bytes
LD DE,8000             ;source-addr is 8000h (beginning of AROS)
PUSH DE                ;save it too
LD HL,(sysc)5EEA       ;destination-addr is the system con. table
PUSH HL                ;save it too
LD DE,0008             ;length of 8 bytes
PUSH DE                ;save it too
LD DE,0001             ;direction forward
PUSH DE                ;save it too
CALL XFER (6722)       ;copy cartridge bytes (8) to SYSCON Table
INC HL                 ;get AROS present? byte
LD A,(HL)              ;and do it now
CP 02                  ;is it AROS and is it present?
JR Z,0A3E (AROSPRE)    ;Yes - jump to 0A3E (AROSPRE)
LD (HL),00             ;No put 'NO AROS PRESENT' there
LD HL,(sysc)5CBC       ;get SYSCON varabile address
ADD HL,DE              ;step over 13 bytes (over cartridge bytes)
LD D,C0                ;reg-address is C0 (CMD)
LD E,00                ;reg-data is 0
CALL WBSR              ;write bank-switch register (call 'it)
CALL asig (0BD1)       ;call asig (0BD1)
JP NC,0AD4             ;if bit 2 was set in ASIG, jump to 0AD4
LD B,A                 ;put max-bank in 'B'
SET 7,B                ;set bit 7 in B (will be 80 if no expansion banks)
LD (HL),B              ;put into end of sys. con. table
RES 7,B                ;Reset bit 7 in B
INC HL                 ;next byte in sysc. con. table
LD C,FE (dest bank)    ;C=FE (B=max-bank) (B=source-bank)
Push BC                ;save BC
```

AROS BYTES

important!

LD DE,0000

0A3E

0A4C →

```
LD DE,Ø8E7(END)   ;DE=Init entry address in EXROM
PUSH DE           ;save it (source address)
LD DE,ØØØØ        ;DE=ØØØØ
PUSH DE           ;save it (destination address)
LD DE,ØØØ1        ;DE=ØØØ1
PUSH DE           ;save it (length)
PUSH DE           ;save it (direction)
CALL XFER(6722)   ;
LD E,FF           ;E=FF (dest-bank) (home bank)
LD D,A            ;D= max_bank (source_bank)
PUSH DE           ;save it
LD DE,ØØØØ        ;DE=ØØØØ (source_address)
PUSH DE           ;save it
PUSH HL           ;save it (destination-address) (SEF7 SYS. Con. TAB)
LD DE,ØØ16        ;DE=ØØ16 (length)
PUSH DE           ;save it
LD DE,ØØØ1        ;DE=ØØØØ (direction)
PUSH DE           ;save it
CALL XFER (6722)  ;
LD D,(HL)         ;get Expansion bank present byte
LD A,(Ø8E7)       ;get Expansion bank data
CP D              ;is it the same?
JP NZ,ØAC2        ;No-Jump to ØAC2
LD C,FE           ;C=FE (dest_bank)
PUSH BC           ;B=max_bank (source_bank)
LD DE,ØA4C        ;DE=ØA4C (source_address)
PUSH DE           ;save it
LD DE,ØØØØ        ;DE=ØØØØ (destination_address)
PUSH DE           ;save it
LD DE,ØØØ1        ;DE=ØØØ1
PUSH DE           ;save it (length)
PUSH DE           ;save it (direction)
CALL XFER         ;
LD E,FF           ;E=FF (dest_bank) (home bank)
LD D,A            ;D= max_bank (source_bank)
PUSH DE           ;save it
LD DE,ØØØØ        ;DE=ØØØØ (source_address)
```

INIT Cont

```
        PUSH DE          ;save it (source_address)
        PUSH HL          ;save it (dest_address) (sys.con.tab.)
        LD DE,0016       ;DE=0016 (length)
        PUSH DE          ;save it
        LD DE,0001       ;DE=0001 (direction)
        PUSH DE          ;save it
        CALL XFER        ;
        LD D,(HL)        ;get Expansion Bank present
        LD A,(08E7)      ;get Expansion Bank present
        CP D             ;are they the same?
        JP NZ,0AC2       ;No-Jump to 0AC2
        DEC HL           ;Dec to 5EF6 in SYSCON table
        DEC HL           ;Dec to 5EF5 in SYSCON table
        CALL 0A0B        ;Exp. Banks are RAM-Init them
        LD DE,0015       ;DE=15 (Skip over 21 bytes)
        ADD HL,DE        ;update sys.con.table.
        JR 0ACA          ;jump forward
0AC2    LD A,D           ;Put Exp Bank status in 'D'
        AND 0F           ;Set flags
        LD (HL),A        ;Put into sys. con. table
        DEC HL           ;dec down a byte in sys. con. tab.
0AC7 -> CALL 0C1F (Chir)
0ACA    LD D,C0          ;addr_reg is C0 (cmd)
        LD E,01          ;data-reg is 01
        CALL WRSR(635C)  ;and write bank-switch reg
        JP 0A4C          ;and jump
0AB4 -> DEC HL           ;lower sys. con. tab. by one
        LD (HL),80       ;end of table marker (put now)
        CALL rmob (0CFB) ;
0ADA -> RET              ;now Return
```

~~Page 1~~ ~~Page 2~~

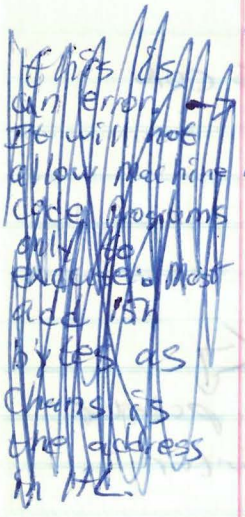
PAGE 3

Init -CONT

```
08F0 -> LD HL,(sysc)(5CBC)   ;get base address of sys con
        LD DE,0008           ;skip over AROS bytes
        ADD HL,DE            ;do it now
        LD A,(HL)            ;get LROS flag
        CP 01                ;is LROS present?
        JR NZ,090F           ;no - jump ->
        PUSH HL              ;yes - and CALL 096C
        CALL 096C            ;save sys con address pointer
0900 -> POP HL               ;get sys con address pointer
        INC HL               ;next byte
        LD E,(HL)            ;get low byte of jump address
        INC HL               ;next byte
        LD D,(HL)            ;get high byte of jump address
        PUSH DE              ;save it (dest_address)
        LD B,00              ;Clear B
        INC HL               ;next byte
        LD C,(HL)            ;get chunk status
        PUSH BC              ;save it (cartridge/chunk select)
        EI                   ;Enable interrupts now
        CALL CBAN (6572)     ;and Goto LROS address ->
090F    LD HL,(sysc)(5CBC)   ;get address of sys con Table
        INC HL               ;step over language type
        LD A,(HL)            ;get AROS byte
        CP 02                ;is AROS present?
        JR Z,091E            ;yes - jump ->
        CALL 096C            ;No - set up BASIC
        JP 099A              ;and jump forward
091E    DEC HL               ;AROS present
        LD A,(HL)            ;get language type
        CP 02                ;is it BASIC and Machine code?
        JR Z,0956            ;yes - jump forward
        CP 03                ;or machine code only
        JR NZ,096A           ;NO - jump 096A - ERROR
        LD DE,0006           ;DE = 0006
        ADD HL,DE            ;skip over 6 bytes
```

Missing LROS

~~This is an error. It will not allow machine code programs only to execute. Most add 15h bytes as chans is the address in HL.~~



```
LD C,(HL)          ;get low number of bytes to reserve
INC HL             ;next byte
LD B,(HL)          ;get high number of bytes to reserve
LD HL,684Ø         ;chans table
ADD HL,BC          ;skip over BC bytes
EX DE,HL           ;exchange DE and HL
LD HL,684Ø         ;chans table
LDIR               ;copy chans table and reserve specified number
CALL Ø973          ;set up BASIC
LD HL,(sysc)       ;get base address of sys con table
LD DE,ØØØ5         ;DE=ØØØ5
ADD HL,DE          ;skip 5 bytes
LD A,(HL)          ;get Autostart byte
CP ØØ              ;is it non-start?
JR Z,Ø99A          ;yes - jump forward
DEC HL             ;point to ~~high byte of start address~~ Chunk spec.
LD C,(HL)          ;'C' holds chunk spec.
DEC HL             ;point to high byte of start addr
LD D,(HL)          ;'D' hold high byte
DEC HL             ;point to low byte of start addr
LD E,(HL)          ;'E' holds low byte
PUSH DE            ;save it (dest_address)
LD B,ØØ            ;B=ØØ
PUSH BC            ;save it (cart/chunk spec)
EI                 ;Enable interrupts
CALL GBANK(6572)   ;and GOTO AROS Start ——>
Ø956 => CALL Ø96C  ;set up BASIC
LD A,8Ø            ;A=8Ø (128 dec)
LD (cart),A        ;signal AROS present
LD HL,18C6(aros)   ;address of AROS in 16K ROM
PUSH HL            ;save it (dest_address)
LD B,FF            ;B=FF (home_bank)
LD C,ØØ            ;C=ØØ (chunk (enable all) status)
PUSH BC            ;save it
CALL GBAN(6572)    ;and finish in Home Rom
Ø96A RST Ø8H       ;Error:
DEFB 7B            ;S-Missing LROS
```

Machine code only

PAGE 9

Finish BASIC boot-up Routine

```
Ø96C -> LD HL,684Ø           ;CROS present - base address of chans
        LD. DE,ØØ15          ;DE = ØØ15
        ADD HL,DE            ;point to 6855 (skip over chans)
Ø973 -> LD (datadd),HL       ;point datadd after chans
        INC HL               ;now point...
        LD (prog),HL         ;PROG and
        LD (vars),HL         ;Vars to location after that
        LD (HL),80           ;terminator byte
        INC HL               ;next location
        LD (e-line),HL       ;points to e-line
        LD (HL),ØD           ;make e-line have 'enter'...
        INC HL               ;and next byte...
        LD (HL),8Ø           ;as a terminator byte
        INC HL               ;next location
        LD (wkspc),HL        ;workspace here
        LD (stbt),HL         ;stack bottom here
        LD (stnd),HL         ;stack end here also
        XOR A                ;clear A
        LD (crfl),A          ;no cros present
        LD (vmod),A          ;normal video mode
0999 -> RET                  ;finish - Return
```

~~Page~~ ~~Page~~

Init-Cont PAGE 5

```
Ø96C -> See other paper (Basic set-up Routine)

Ø99A -> LD D,FF          ;D=FF
  ↗     LD E,8Ø          ;E=8Ø
Non-    LD HL,ØE2F       ;HL=ØE2F (address of MAIN-1 start of basic)
Start   PUSH HL          ;save it (dest_address)
ARØS    PUSH DE          ;save it (temp)
        LD HL,(sysc)(5CBC) ;SYS CON ADDRESS
        LD DE,ØØØC       ;prepare to skip
        ADD HL,DE        ;over cartridge bytes
        LD B,ØØ          ;Clear 'B'
Ø9AC -> LD A,(HL)        ;get terminator byte
        CP 8Ø            ;is it the terminator byte
        JR Z,ØAE3        ;yes jump forward ->
        CP ØØ            ;is it ØØ No exp banks?
        JR Z,Ø9DD        ;yes jump forward ->
        INC HL           ;next byte
        LD B,(HL)        ;get presence flag
        LD DE,ØØ14       ;DE=ØØ14
        ADD HL,DE        ;skip over 2Ø bytes
        LD A,(HL)        ;get device type
        RRCA             ;is it Initializable?
        JR C,Ø9C4        ;yes - jump ->
        INC HL           ;No -
        INC HL           ;-↓
        INC HL           ;Point to next device specification
        JR Ø9AC          ;jump back
Ø9C4 -> INC HL           ;point to device boot prioty
  ↗     LD A,(HL)        ;get boot priority
Initializable POP DE     ;get DE
        CP E             ;~~is it 8Ø~~ (or below 8Øh)
        JR C,Ø9CF        ;yes - boot of Ø to 7F
        PUSH DE          ;NO - resave DE
        INC HL           ;Point to...
        INC HL           ;next device specification
```

```
                 128 64 32 16 8 4 2 1
58 = 0 0 1 1 1 0 1 0
     ↑ ↑       ↑   ↑
     on on     on  on
```

```
                JR 09AC          ;and now retry
09CF ->         POP DE           ;get DE
↑               LD DE,0005       ;DE=5
good boot       SBC HL,DE        ;sub HL-DE (-5 bytes)
                PUSH HL          ;save HL
                LD C,A           ;put A into 'C' (device boot)
                PUSH BC          ;save BC
JR 09AC         INC DE           ;INC DE
                INC DE           ;INC DE (DE=7)
                ADD HL,DE        ;point to next device specifiation
09DD ->         LD DE,0018       ;DE=18
↑               ADD HL,DE        ;point to next device specification
NO Exp          JR 09AC          ;try again
Banks.

09E3 ->         POP BC           ;Put DE into BC
↑               LD A,B           ;get B (which is FF)
term            CP FF            ;is it ff?
byte            JR Z,09ED        ;yes - jump forward
09E9 ->         LD C,58          ;C=58 (chunk spec)
                JR 09EF          ;B=FF (home_bank) and jmp forward
09ED ->         LD C,00          ;C=00 (chunk all on) ;(B=home_bank)
09EF ->         PUSH BC          ;save it
                EI               ;enable interrupts      Boot up Basic ---
                CALL GBANK(6512) ;and jump to 0E2F in Home ROM
09F4 ->         ↙ See sheet 1 ←
```

NRAM Subroutine

```
ØADB  LD (HL), Ø2          ; Put Ø2 in SYS. CON. TABEL (Ø2 - RAM present)
      PUSH BC (B=max_bank, C=EXROM) ; Save max_banks and chunk select
      LD DE, ØØ38          ; DE=ØØ38 (source_address) INT...
      PUSH DE              ; save it FIELDER
      PUSH DE              ; save it (dest_address) INT FIELDER
      LD DE, ØØ1Ø          ; DE=ØØ1Ø (length)
      PUSH DE              ; save it
      LD DE, ØØØ1          ; DE=ØØØ1 (direction)
      PUSH DE              ; save it
      CALL XFER (E722)     ;
      INC HL               ; SYSCON TO 5EF6
      INC HL               ; SYSCON TO 5EF7
      LD A, (HL)           ; get Expansion bank info
      SET Ø, A             ; bank must be here (bank number)
      LD (HL), A           ; put it into syscon table
      LD DE, ØØØØ         ; Clear DE
      LD A, Ø1             ; A = Ø1
ØAF9  EX AF, AF'           ; put into prime
      PUSH HL              ; save syscon table pointer
      EX DE, HL            ; switch regs
      LD DE, 2ØØØ          ; DE = 2ØØØ
      ADD HL, DE           ; HL = 2ØØØ
      EX DE, HL            ; switch REGS
      POP HL               ; restore pointer
      LD B, FE             ; B = FE (dest_bank)
      LD A, (maxb)(5CBE)   ; get max_banks
      LD C, A              ; C = max_banks (source bank)
      PUSH BC              ; save it
      LD BC, 08E7          ; BC=Ø8E7 (source_address)
      PUSH BC              ; Save it
      PUSH DE              ; save it (dest_address) (2000h?)
      LD BC, ØØØ1          ; BC=ØØØ1 (length)
      PUSH BC              ; save it
      PUSH BC              ; save it (direction)
      CALL XFER            ;
```

```
                LD  A,(maxb)(5CBE)      ;get max-bank
                LD  B,A                 ;store in 'B' (source_bank)
                LD  C,00                ; C = 00 (dest_bank)
                PUSH BC                 ;save it
                PUSH DE                 ;save it (source address)(2000h?)
                INC HL                  ;update sys. con. fig. tab. pointer
                PUSH HL                 ;save it (dest_address)
                LD  BC,0001             ; BC = 0001 (length)
                Push BC                 ; save it  (length)
                Push BC                 ; Save it  (direction)
                CALL XFER               ;
0B38 ->         LD  B,(HL)              ; get Expansion Bank Status
                DEC HL                  ; lower SYSCON Table one
                LD  A,(0BE7)            ; get Expansion Bank Status
                CP  B                   ; are they the same?
                JR  NZ,0B95             ; No - Jup to 0B95
                LD  B,FE                ; B = FE (source_bank)
                LD  A,(maxb)(5CBE)      ; get max_bank
                LD  C,A                 ; C = max_bank (dest_bank)
                PUSH BC                 ;save it
                LD  BC,0A4C             ; BC = (source_address)
                PUSH BC                 ; save it
                Push DE                 ; Save it (dest-addr) (2000h?)
                LD  BC,0001             ; BC = (length)
                PUSH BC                 ; save it
                PUSH BC                 ; save it (direction)
                CALL XFER (0722)        ;
                LD  A,(maxb)(5CBE)      ;get max_bank
                LD  B,A                 ; B = max_bank (source_bank)
                LD  C,00                ; C = cart_bank (dest_bank)
                PUSH BC                 ; save it
                PUSH DE                 ; save it (source_addr) (2000h?)
                INC HL                  ;update sys. con. table pointer
                PUSH HL                 ; (HL = dest_address)
                LD  BC,0001             ; BC = 0001 (length)
                Push BC                 ; save it
                Push BC                 ;save it (direction)
```

7 CONT —

Page

```
        CALL XFRR (6722)        ;
        LD B,(HL)               ;get Exp. Bank status
        DEC HL                  ;dec sys. con. table addr
        LD A,(ØA4C)             ;get Exp. Bank status
        CP B                    ;do they match
        JR NZ,ØB95              ;No - Jump to ØB95
        EX AF,AF'               ;Yes - a match - switch to primes
        LD B,(HL)               ;get Exp. Bank status
        CP Ø1                   ;is it a 1?
        JR NZ,ØB68              ;No - jump to ØB68
        SET 1,B                 ;yes - cont. Set 1,B
        JR ØB92                 ;and jump to ØB92
ØB68    CP Ø2                   ;is it Ø2?
        JR NZ,ØB7Ø              ;No - jump to ØB7Ø
        Set 2,B                 ;yse - set 2,B
        JR ØB92                 ;jump ->
ØB7Ø    CP Ø3                   ;is it 3?
        JR NZ,ØB78              ;no - jump ->
        SET 3,B                 ;yes -
        JR ØB92                 ;jump
ØB78    CP Ø4                   ;is it 4?
        JR NZ,ØB8Ø              ;no - jump ->
        SET 4,B                 ;yes
        JR ØB92                 ;jump ->
ØB8Ø    CP Ø5                   ;is it 5?
        JR NZ,ØB88              ;No - jump ->
        SET 5,B                 ;yes
        JR ØB92                 ;jump ->
ØB88    CP Ø6                   ;is it 6?
        JR NZ,ØB9Ø              ;no - jump ->
        SET 6,B                 ;yes
        JR ØB92                 ;jump ->
ØB9Ø    SET 7,B                 ;must be 7
ØB92    LD (HL),B               ;save Exp bank status
        JR ØBCA                 ;and jump ->
```

```
                                  ↓ Also set old value (Ø1?)
ØB95    Ex AF,AF'                 ;switch AF to primes'
        LD B,(HL)                 ;get Exp. Bank stat
        CP Ø1                     ;is it Ø1?
        JR NZ,ØB9F                ;NO -jump
        RES 1,B                   ;yes
        JR ØBC9                   ;jump->
ØB9F    CP Ø2                     ;is it 2
        JR NZ,ØBA7                ;NO -jump
        RES 2,B                   ;yes
        JR ØBC9                   ;jump->
ØBA7    CP Ø3                     ;is it 3
        JR NZ,ØBAF                ;NO -jump
        RES 3,B                   ;yes
        JR ØBC9                   ;jump
ØBAF    CP Ø4                     ;is it 4
        JR NZ,ØBB7                ;NO -jump
        RES 4,B                   ;yes
        JR ØBC9                   ;jump
ØBB7    CP Ø5                     ;is it 5?
        JR NZ,ØBBF                ;NO -jump ->
        RES 5,B                   ;yes
        JR ØBC9                   ;jump
ØBBF    CP Ø6                     ;is it 6
        JR NZ,ØBC7                ;NO -jump
        RES 6,B                   ;yes
        JR ØBC9                   ;jump
ØBC7    RES 7,B                   ;must be 7
ØBC9    LD (HL),B                 ;save Exp. Bank status
        INC A                     ;Ing Exp. Bank status
        CP Ø8                     ;is it 8?
        JP NZ,ØAF9                ;NO -jump
ØBDØ ->  RET                       ;yes Return
```

ASIG

```
ØBD1  LD A,(maxb) (5CBE)    ;get number of expansion banks
      INC A                 ;inc it by one
      LD (maxb),A (5CBE)    ;set maxbank to new number
      LD (6315),A           ;also the bank-switch maxbank
      LD D,AØ               ;reg address is AØ (ABN)
      LD E,A                ;reg-data is max_bank
      CALL WBSR(635C)       ;and write bank-switch reg
      LD D,8Ø               ;reg-address is 8Ø (BNA)
      LD E,A                ;reg-data is max-bank
      CALL WBSR(635C)       ;and write bank-switch reg
      LD D,4Ø               ;reg-addres is 40 (HS)
      LD E,ØØ               ;reg-data is 0Ø
      CALL WBSR(635C)       ;and write bank-switch reg
      PUSH AF               ;save max-bank status
      LD A,(AØØØ)           ;get contents of (AØØØ)
      EX AF,AF1             ;put it into the prime
      LD A,Ø4               ;A=4...
      LD (AØØØ),A           ;Put a 4 into (AØØØ)
      LD D,AØ               ;LSN-address is AØ  (ABN)
      LD E,CØ               ;MSN-address is cØ  (cmD)
      CALL RBSR(63AD)       ;and Read the bank-switch regs
      BIT 2,E               ;IS E '4' or Bit 2 set?
      JR NZ,ØCØA            ;Yes-goto ØCØA
      EX AF,AF'             ;get old contents of (AØØØ)
      LD (AØØØ),A           ;restore it
      POP AF                ;get old max_bank status
      SCF                   ;set carry flag (Expansion banks present)
      RET                   ;Return to calling routine
ØCØA  EX AF,AF'             ;get old contents of (AØØØ)
No Expansion banks
      LD (AØØØ),A           ;put old contents back
      POP AF                ;get bank_max status
      DEC A                 ;dec it by one
      LD (maxb),A           ;Put into variables
      LD (6315),A           ;
```

```
                LD D,C0 (cmD)       ; add-reg is C0 (cmD)
                LD E,04             ; data-reg is 04
                CALL WBSR (635C)    ; and write bank switch reg
                AND A               ; clear carry flag
0C1E ->         RET                 ; and Return
```

PAGE 9

Chir Routine

```
0C1F   DEC HL          ;point one byte lower IN sys con table
       LD (HL),01      ;put 1 there
       LD DE,0015      ;DE=15
       ADD HL,DE       ;skip over 15h bytes
       LD A,(HL)       ;get this byte
       RRA             ;and see if it is 1
       JR C,cidi(0C2F) ;yes-jump forward
       LD DE,0004      ;DE=4
       ADD HL,DE       ;point to next entry
       RET             ;and return
cidi   LD C,08         ;C=08
       LD A,(maxb)(5CBE) ;get max_banks
       ~~LD D,(HL)~~   ;
       DEC HL          ;
       DEC HL          ;
       DEC HL          ;point to device Reset addr
       LD D,(HL)       ;get high byte
       DEC HL          ;point to low byte
       LD E,(HL)       ;get low byte
       LD H,D          ;Put into the
       LD L,E          ;HL pair
       LD B,A          ;max_banks in B
       PUSH HL         ;save it (dest_addr)
       PUSH BC         ;save it (bank/chunk)
       LD BC,0000      ;;Clear BC
       PUSH BC         ;save it (no parms)
       PUSH BC         ;save it (no parms)
       CALL CBNK(65D0) ;Reset bank
       LD DE,0008      ;DE=08
       ADD HL,DE       ;skip over 8 bytes
0C4B   RET             ;and return
```

RSET Subroutine

```
0C4C -> XOR A              ;clear 'A'
        LD (maxh),A (5CBE) ; No Banks
        LD (6315),A        ;
        LD D,C0            ; D=C0 (CMD)
        LD E,00            ; E=00 (Reset)
        CALL NBSR (035C)   ; and write to the reg
        LD HL,(sysc) (5CBC) ;set base of table
        LD DE,0006         ;skip over...
        ADD HL,DE          ;the cartridge bytes
        CALL asig (0BD1)   ;
        JP NC,0CF5         ; No Exp banks -jump
        LD A,(HL)          ;get term byte
        PUSH HL            ;save it
        CP 80              ;is it the end-of-table?
        JR NZ,0C72         ; No -jump forward
        LD DE,0018         ;LD DE,0018
        ADD HL,DE          ;skip over 24 bytes
        LD (HL),A          ;put 'A' there
        CALL asig (0BD1)
        LD HL,5FE9         ;HL points to here
        LD E,FF            ; E=FF (dest_bank)
        LD D,00            ; D=00 (cart_bank)
        PUSH DE            ;save it
        LD DE,0000         ;DE=(source_address)
        PUSH DE            ;save it
        PUSH HL            ;save it (dest_address) SYS CON Table
        LD DE,0016         ;length of 22 bytes
        PUSH DE            ;save it
        LD DE,0001         ;direction forward
        PUSH DE            ;save it
        CALL XFER (0792)   ;set Exp bank Status
        EX AF,AF'          ;save term byte
        LD A,(HL)          ;get Exp bank byte
        CPL                ;complement it
        DEC HL             ;point to ~~bank number~~ next byte
```

```
LD (HL),A          ;put chuck there
EX AF,AF'          ;get term byte (bank number)
LD D,FF            ;home bank (~~FF~~ source bank)
LD E,A             ;(Exp bank is dest-bank)
~~SAVE~~ PUSH DE   ;save it
PUSH HL            ;save it (source_address)
LD DE,0002         ;02 = (dest address)
PUSH DE            ;save it
LD DE,0001         ;01 = (length)
PUSH DE            ;save it
PUSH DE            ;save it (direction)
CALL XFER (6722)
LD E,FF            ;(dest-bank = home bank)
LD D,A             ;(source bank = Exp bank)
PUSH DE            ;save it
LD DE,0002         ;source_address
PUSH DE            ;save it
INC HL             ;inc SYS CON TABLE
PUSH HL            ;save it (dest_address)
LD DE,0001         ;DE=01
PUSH DE            ;save it (length)
PUSH DE            ;save it (direction)
CALL XFER (6722)
LD A,(HL)          ;get this byte
DEC HL             ;dec pointer
LD B,(HL)          ;get this byte
CP B               ;are they the same?
JR NZ,0CCA         ;NO, jump forward
POP HL             ;yes-get SYS CON address
LD A,(HL)          ;get byte there
CP 02              ;is it 2 (RAM?)
JR NZ,0CC5         ;NO - jump forward
INC HL             ;yes - point to
INC HL             ;chunk spec
JR 0CE8            ;jump forward
0CC5-> Call Nram (0ADB)
```

~~Page 2~~ PAGE 11

RSET Cont

```
0CC8 -> JR 0CE8          ;and jump forward
0CCA -> LD C,(HL)        ;set byte there
        POP HL           ;get sys con address
        INC HL           ;Point to...
        INC HL           ;device spec
        LD A,(HL)        ;set byte there
        CP C             ;are they the same?
        JR Z,~~0CE8~~ 0CE8  ;yes-jump
        PUSH HL          ;NO - save pointer
        EX DE,HL         ;change regs (~~source at~~ (dest addr)
        LD HL,5FE9       ;point here (~~dest addr~~ (source addr)
        LD BC,~~16~~ 0016  ;BC= 22 bytes (length)
        LDIR             ;copy into SYS CON Table
        POP HL           ;get SYS CON Table
        DEC HL           ;repoint pointer
        LD A,(maxb)(5CBE)  ;get maxbank
        SET 7,A          ;make it 8X (x=bank)
        LD (HL),A        ;Put into (bank-number)
        CALL chir (0C1F)  ;
        INC HL           ;point to device spec
        LD D,C0          ;C0 = (CMD)
        LD E,01          ;01 =(Intellegent device?)
        CALL WPSR (635C)  ;
        LD DE,0016       ;prepare to skip over...
        JP 0D60          ;22 bytes and jump back
0CF5 -> LD (HL),80       ;put term byte here
        CALL rnob (0CFB)  ;
0CFA -> RET              ;and return
```

RNOB Subroutine

```
ØCFB ->  XOR A              ; clear 'a'
         LD (maxb),A (5CBE) ; set max_banks to Ø (none!)
ØCFF ->  LD HL,(sysc) (5CBC) ; get base address of Sys Con. tab.
         LD DE,ØØØC         ; DE=ØØØC
         ADD HL,DE          ; repoint to 5EF6 (after cartridge bytes)
ØDØ6 ->  LD A,(HL)          ; Get byte there
         CP 8Ø              ; is it the terminator?
         JR Z,ØD84          ; yes - jump
         INC HL (HL=5EF7)   ; NO - Expansion banks present (step over term byte)
         LD A,(HL)          ; get byte there
         Bit Z,A            ; is bit 7 set?
         JR NZ,ØD17         ; yes - jump
         LD DE,ØØ17         ; NO - repoint sys con table pointer
         ADD HL,DE          ; to (5EFØE) or more until a match is found
         JR ØD Ø6           ; and redo again
ØD17     LD (5FE9),HL       ; put the sys con tab. address here
         DEC HL             ; check term byte
         LD A,(HL)          ; get that byte
         CP Ø2              ; is it 2 (Ram Expansion)
         JR NZ,ØD28         ; NO - jump
         LD DE,ØØ17         ; yes - DE=ØØ17
         ADD HL,DE          ; repoint sys con pointer
         LD A,FF            ; A=FF
         JR ØD2D            ; jump forward
ØD28     LD DE,ØØ17         ; Not RAM Expansion - DE=ØØ17
         ADD HL,DE          ; repoint sys con pointer
         LD A,(HL)          ; get byte at new point
ØD2D ->  [illegible] LD (5FEB),A ; put [illegible] byte here
ØD3Ø -   INC HL             ; next byte
         LD A,(HL)          ; get byte
         CP 8Ø              ; is it end of table?
         JR Z,ØD64          ; yes - jump
         INC HL             ; No -
         LD A,(HL)          ; Next byte (presense flag)
         BET 7,A            ; is bit 7 set?
```

NOTE: If this routine is entered without the correct SYS CON Table, It will Crash the machine

```
        JR NZ,ØØ42      ;yes - jump forward
        LD DE,ØØ17      ;No - Get next Exp bank
        JR ØØ3Ø         ;retry again
ØØ42    DEC HL          ;back one byte (term byte)
        LD A,(HL)       ;get byte there (presence flag)
        CP Ø2           ;is it RAM Expansion?
        JR NZ,ØØ4E      ;No - jump forward
        LD DE,ØØ17      ;yes - DE=ØØ17
        ADD HL,DE       ;update SYS CON table
        JR ØØ3Ø         ;jump backward and retry
ØØ4E    EX DE,HL        ;change regs (DE holds SYS CON)
        LD BC,ØØ17      ;BC=ØØ17
        ADD HL,BC       ;basicly add 17+17 (=2E)
        LD A,(5FEB)     ;get an old SYS CON address
        LD B,A          ;put into 'B' high byte
        LD A,(HL)       ;get byte at (HL) not SYS CON
        CP B            ;is it the same
        JR NC,ØØ3Ø      ;bad - rescan (retry)
        LD (5FEB),A     ;good - store these
        LD (5FE9),DE    ;Regs in SYS CON table
        JR ØØ3Ø         ;Retry now!!
ØØ64    LD A,(5CBE) (maxb)  ;get max bank
        INC A           ;inc by one
        LD (maxb),A     ;place in maxbank and...
        LD HL,(5FE9)    ;get this address (sys con)
        LD (HL),A       ;put max_bank here ØØ6Ø
        LD E,FF         ;dest_bank = home bank
        LD D,A          ;source_bank = Exp bank
        PUSH DE         ;save it
        LD E,(HL)       ;get lower byte
        LD D,ØØ         ;high byte is Ø
        PUSH DE         ;save it (source_address)
        LD E,ØØ         ;Now E=ØØ
        PUSH DE         ;save it (dest_address)
        LD E,Ø1         ;E now = 1
        PUSH DE         ;save it (length)
        PUSH DE         ;save it (direction)
```

~~PAGE 2~~ RNOB Cont—

PAGE 13

```
        CALL XFER                ;
        JP ØCFF                  ; jump forward
ØD84    XOR A                    ; here for AROS cart
        LD (maxb), A (5CBE)      ; no max_banks
        LD (6315), A             ; also in BS code
        LD D, CØ                 ; add_reg = (CMD)
        LD E, ØØ                 ; data_reg = ØØ
        CALL WBSR (635C)         ; and write bank_switch register
        LD HL, (sysc)            ; get address for table
        LD DE, ØØØD              ; DE = ØØØD
        ADD HL, DE               ; skip cartridge bytes
        LD D, AØ                 ; D = AØ (note - this instruction not needed)
ØD9B -> CALL ØBD1 (asig)         ; assign Exp Banks - carry if banks / No carry - no banks
        RET NC                   ; Return as no Exp banks exsist
        LD E, (HL)               ; banks do exsist (get byte)
        CALL WBSR (635C)         ; write bank_switch regs
        LD D, CØ                 ; D = CØ or (CMD) command
        LD E, Ø1                 ; E = Ø1
        CALL WBSR (635C)         ; write to bank_switch regs
        LD DE, ØØ18              ; DE = ØØ18
        ADD HL, DE               ; skip over first exp bank
ØDAE -> JP ØD9B                  ; and jump backward
```

—> End of INIT

Change Video routine follows this

Beginning of EXROM Boot UP
(if power comes on in EXROM bank)

IN EXROM!

```
0000    DI              ;Disable interrupts
        JR 0049         ;jump ahead
        FF,FF,FF,FF,FF  ;unused bytes
0008    LD HL,(chad)(5C5D)  ;address reached by the
Restart Error
        LD (xptr),HL (5C5E) ;interpreter is copied to the error
        POP HL          ;pointer before proceeding. Get address of error code
        LD C,(HL)       ;get error code in 'C'
        LD (ernr),L (5C3A)  ;put into error number variable
        LD SP,(err-sp) (5C3D) ;make SP point to error calling address
        LD HL,sest (1354)   ;goto 1354 in home ROM (set-stk)
        PUSH HL         ;save it (dest address)
        LD H,FF         ;FF=home bank
        LD L,00         ;00=chunk selection
        PUSH HL         ;save it
        LD A,(vidmode)(5CC2)  ;get video mode
        AND A           ;is it 0?
        NOP             ;wait 4T states
        JR Z,002C       ;normal video mode is set
        POP AF          ;get 'HL' bank/chunk selection
        CALL FD32       ;goto FD32 (video mode not normal)
002C    POP AF          ;get 'HL' bank/chunk selection
        CALL GBAK (0572)  ;and goto Home rom -->
0030    FF,FF,FF,FF,FF,FF,FF,FF  ;unused bytes
0038    PUSH AF         ;save AF
INT     DI              ;Disable interrupts (not needed)
CALL    LD A,(vmode)(5CC2)  ;get video mode
        AND A           ;is it 0? (normal mode set)
        NOP             ;wait 4T states
        JR Z,0045       ;yes normal video - jump to 0045
        POP AF          ;restore AF (not normal video)
        JP FA6E         ;in Chunk 7, finish there -->
0045    POP AF          ;restore AF (normal video)
        JP 62AE         ;jump to Chunk3, normal video finish->
0049    LD A,01         ;system init. continues here
```

Note! RST 00h, RST 18h, RST 20h, RST 28h, and RST 30h cannot be used

Which goes to Home ROM

```
                OUT (F4),A      ;enable only chunk 0 in EXROM bank
                JR 005A         ;jump forward
0040 ->         XOR A           ;clear A
                OUT (F4),A      ;enable all of home bank
                OUT (FF),A      ;normal video/keyboard INT/No EXROM
                LD DE,FFFF      ;Set top of RAM (65535)
                JP init (0D31)  ;jump to init in home ROM
005A            LD HL,004F      ;source is 004F
                LD DE,6000      ;destination is 6000h (RAM)
                LD BC,000B      ;length is 12 bytes
                LDDR            ;Copy now
                JP 6000         ;Finish through 6000h (see above)
```

This is Copied into Ram and Executed

0068 -> thru 08A9 -> } Cassette Handling Routines

Note: These cassette Routines are largely the same as the ZX-Spectrum Cassette routines. See the 'Cassette Routines' additions and Changes for more information

Added Routines:

```
08AA            PUSH AF         }
                PUSH BC         } save registers
                PUSH DE         }
WAIT KEY        LD BC,04C0      ;length of delay loop
ROUTINE
08B0 —          DEC BC          }
                LD A,C          } this is the actual delay loop
                OR B            }
                JR NZ,08B0      }
08B5 ->         XOR A           ;clear A
                IN A,(FE)       ;get keyboard state
                AND 1F          ;get only keyboard bits
                CP 1F           ;are no keys pressed?
                JR Z,08B5       ;yes-rescan
                PUSH EX         ;a key is pressed (Save EX)
                EXX             ;use prime regs
                LD HC,08A9      ;ROM address (CLS-LOWER) IN HOME ROM
```

EXROM Cont -

```
        PUSH HL          ;save it (dest-addr)
        LD L,ØØ          ;ØØ = enable all of home-bank
        LD H,FF          ;FF = bank home bank
        PUSH HL          ;save it (bank/chunk)
        LD HL,ØØØØ       ;No parms
        PUSH HL          ;no parms
        PUSH HL          ;no parms
        EXX              ;replace regs
        CALL cbnk (ØF99) ;and jump forward
        POP IX           }
        POP DE           } Restore Regs
        POP BC           }
        POP AF           }
        RET              and return
Ø8D9 -> EXX              ;use prime regs
syer ↗  LD HL,1BED       ;error C in Home ROM address
        PUSH HL          ;save it (dest-address)
error   LD L,ØØ          ;L=ØØ (enable all of home-bank)
C ROUTINE LD H,FF        ;H=FF (home-bank)
        PUSH HL          ;save it (bank/chunks)
        EXX              ;restore regs
        CALL Jbnk (ØF8A) ;and jump to ROM with no return.

Ø8E7    }
thru    } Iniazation Routines, see other sheets
9DAF    } for dissassembly

ØDBØ -> PUSH BC          }
second  PUSH DE          } SAVE Regs
Disply file PUSH HL      }
Routine PUSH AF          }
        LD HL,(pram)(5CB4)  ;get Top of RAM
        LD DE,(uds-)(5C7B)  ;sed address of 'UDG-'
        AND A            ;clear cary flag
        SBC HL,DE        ;sub HL-DE (get length of UDG-)
```

Note!!!
Must call only through CALL.BANK Routine or this Routine will Crash Machine

```
                LD B,IY          } Put result into BC
                LD C,L           }   (length)
                INC BC           ; inc by one
                LD HL,(udg-)     ; get (UDG) address again
                PUSH HL          ; save it
                LD DE,0840       ; DE=840h (2112 bytes in dec)
                AND A            ; clear carry
                SBC HL,DE        ; find new address (dest_address)
                EX DE,HL         ; change regs
                POP HL           ; get old UDG- (source_addr)
                LD (udg-),DE     ; put new value in it now
                LDIR             ; copy UDG- to new location
                LD HL,0000       ; clear HL
                ADD HL,SP        ; HL now holds SP
                LD BC,97C0       ; BC=97C0
                ADD HL,BC        ; add SP+97C0 (38848 in dec)
                DI               ; disable Interrups
                LD SP,HL         ; relocate Machine stack
                LD DE,F7C0       ; destination address
0DE0 ->         LD HL,6000       ; source address
                LD BC,0840       ; length (2112 bytes in dec)
                LDIR             ; copy RAM-RES code to Chunk 7
                LD HL,1000       ; source address
                LD BC,97C0       ; BC=97C0
0DEE ->         LD E,(HL)        ; get first byte
                INC HL           ; next byte
                LD D,(HL)        ; get next byte
                INC HL           ; next byte
                LD A,E           ; put E into A
                OR D             ; and 'OR' it with D
                JR Z,0E05        ; are they 0? Yes jump  Table finished?
                EX DE,HL         ; switch Regs
                ADD HL,BC        ; point to Chunk 7
0DF8 ->         PUSH DE          ; save address
                LD E,(HL)        ; set next byte
                INC HL           ; update HL
```

Transfer Fixup Table to Chunk 7

PAGE 16 EXROM Cont

```
        LD D,(HL)          ;get this byte
        EX DE,HL           ;swiths Regs (HL=64Ø5)
        ADD HL,BC          ;update HL
        EX DE,HL           ;switch again
        LD (HL),D          ;put D there
        DEC HL             ;update HL
        LD (HL),E          ;put E there
        POP HL             ;get old address from DE
        JR ØDEE            ;and jump ahead
ØEØ5    POP AF             ;clean flags (NOTE: Machine stack has been relocated)
        LD (vdmode),A      ;put A into video mode
        PUSH AF            ;save AF
        EI                 ;Enable interrupts
        LD HL,6ØØØ         ;HL=6ØØØ
ØEØE -> XOR A              ;clear 'a'
        LD (HL),A          ;put into (HL) (new display file)
        INC HL             ;inc address by one
        LD A,H             ;Put high address in 'A'
        CP 7B              ;is it 7B (finished clearing the display)
        JR NZ,ØEØE         ;No - do again
        POP AF             ;get AF
        PUSH AF            ;save it
Error -> AND 7F            ;get only bits 6-Ø
Should  LD B,A             ;put into B
be      IN A,(FF)          ;get display port
AND 3F  AND 8Ø             ;get EXROM bit (leave alone)
to get  OR B               ;merge B with A
only bits OUT (FF),A       ;effect mode change
5-Ø     POP HL             }
        POP DE             }  Pop Restore  (Pop off of Bank Switching Stack!!!)
        POP BC             }  note machine stack has
        POP AF             }  been relocated
        RET                ;and Return
```

Note only call through CALL-BANK Routine as a direct call will crash SYSTEM

```
ØE27  PUSH AF
Close PUSH BC
Disfile PUSH DE
#2    PUSH HL                  } SAVE REGS
error IN A,(FF)                ;get video mode
Should AND 8Ø                  ; Clear bits 6-Ø
be    OUT (FF),A               ;set mode to Ø
~~AND 3F~~ LD HL,ØØØØ              ;Clear HL
      ADD HL,SP                ;HL now is also SP
      LD DE,97CØ               ;DE=97CØ (38848 dec)
      AND A                    ;Clear carry
AND CØ SBC HL,DE               ;reset stack pointer
      DI                       ; Disable interrupts
      LD SP,HL                 ;now reset SP
      LD HL,FFFF               ; HL=source-addr
      LD DE,683F               ; DE=destaddr
      LD BC,Ø84Ø               ; BC=length
      LDDR                     ; And copy RAM-RES to Chunk 3
      LD HL,1DØØ               ;Base address of Fixtable
      LD BC,97CØ               ;BC=length between Chunk 3 and 7
ØE4E  LD E,(HL)                ;get first entry
      INC HL                   ; Next entry
      LD D,(HL)                ;get it too
      INC HL                   ;Next entry
      LD A,E                   ;Put entry into A
      OR D                     ;is it a double ØØ?
      JR Z,ØE66                ;yes we are finished with table
      PUSH HL                  ;save address
      EX DE,HL                 ;switch Regs
      LD E,(HL)                ;Get byte at called addr
      INC HL                   ;next byte of call addr
      LD D,(HL)                ;get this byte to
      ~~SW~~ EX DE,HL             ;switch regs
      ~~CL~~ AND A                ;clear flags
      SBC HL,BC                ;new address here
      EX DE,HL                 ;switch regs
      ~~NOW~~ LD (HL),D           ;replace with corrected
                                 value
```

EXROM CONT

```
0E61 -> DEC HL          ;point to low byte
        LD (HL),E       ;correct this too
        POP HL          ;get table address
        JR 0E4E         ;and continue until end of table
0E66 -> XOR A           ;Clear A reg
        LD (vmod),A(5C8A)  ;reset video mode
        EI              ;enable interrupts
        LD HL,F7BF      ;HL= source address
        LD DE,FFFF      ;DE = dest_address
        PUSH HL         ;save address
        LD BC,(uds-)    ;get address of UDG-
        AND A           ;clear carry
        SBC HL,BC       ;get length
        LD B,H          ;and transfer
        LD C,L          ;to BC
        INC BC          ;inc length by one
        POP HL          ;restore source address
        LDDR            ;and copy UDG- back to high RAM
0E7F -> LD DE,0840      ;get difference
        LD HL,(uds-)    ;HL = base of UDG-
        ADD HL,DE       ;update to new address
        LD (UDG-),HL    ;and put into UDG-
        POP HL          }
        POP DE          } Restore Regs
        POP BC          }
        POP AF          }
        RET             ;and return

0E8E    PUSH BC         }
svid -> PUSH DE         } SAVE Regs
        PUSH HL         }
        PUSH AF         }
        LD B,A          ; save A (Video mode)
        LD A,(vmod)     ;get video mode
        AND A           ;set/reset zero flag
```

```
                JR NZ,0EED      ; not mode 0 - jump to 0EED
                OR B            ; mode = 0 - merge A and B
                JP Z,0F3D       ; mode = 0 - jump 0F3D
                LD HL,12C0      ; mode <> 0 | HL = 12C0 (length)
                LD B,H          ; transfer to
                LD C,L          ; BC (length)
                LD DE,0840      ; DE = length of RAM-RES
                ADD HL,DE       ; update HL+DE
                LD DE,(stend) (5C65)  ; get stk end
                ADD HL,DE       ; also add to HL
                LD DE,(ramtop) (5CB2) ; get RAMTOP
                AND A           ; clear carry
                SBC HL,DE       ; check for free memory
                JP NC,0F3A      ; Not enough memory - jump 0F3A
                LD HL,683F      ; yes - enough memory
                LD DE,12CA      ; DE = 12CA
                PUSH DE         ; save it (dest-address)
                LD DE,FF00      ; DE = FF00
0FEBF ->        PUSH DE         ; save it (bank/chunk)
                LD DE,0000      ; DE = 0000
                PUSH DE         ; no parms to routine
                PUSH DE         ; no parms from patent
                CALL CBAN (6500)  ; GoTo 12CA in Home ROM (clean up SYS. varib.)
                LD HL,(stend)   ; HL = used memory
                EX DE,HL        ; put into DE
                LDDR            ; [illegible]
                POP AF          ; get mode from entry
                PUSH AF         ; save it again
                CALL opd2 (0DB0)  ; open second display file
                LD BC,97C0      ; length between Chunk 3 and 7
                LD HL,(ersp) (5C3D)  ; get address of error SP
                ADD HL,BC       ; update error SP
                LD (ersp),HL    ; and enforce it now
                LD HL,(lisp) (5C3F)  }
                ADD HL,BC       }  also List SP
                LD (lisp),HL    }
                LD HL,(msth) (5CC0)  ; get address above machine stack
```

EXROM CONT

```
ØEE7   ADD HL,BC          } and update it too
       LD (mstb),HL
       JR ØF3D            ; and jump forward
ØEED   LD A,B             ; get mode from entry
       AND A              ; clear carry (check if zero)
       JR Z,ØFØ1          ; mode=Ø ; jump->
error -> AND 7F           ; mode <> Ø ; get bits 6-Ø
       LD B,A             ; put into B
       IN A,(FF)          ; get current video mode
       AND 8Ø             ; get bit 7
       OR B               ; mask with video mode
       OUT (FF),A         ; and change video mode
       LD A,B             ; get video mode
       LD (vmod),A        ; and set (Video mode) to it
       JR ØF3D            ; and jump forward
ØFØ1 -> CALL cld2 (ØE27)  ; close display file 2
       LD BC,97CØ         ; length between Chunk 3 and 7
       LD HL,(ersp)       ; get error SP
       AND A              ; clear carry
       SBC HL,BC          ; and update to new position
       LD (ersp),HL       ; and enforce it
       LD HL,(lisp)       }
       AND A              } and with Lisp SP too
       SBC HL,BC          }
       LD (lisp),HL       }
       LD HL,(mstb)       }
       AND A              } and with Mashine stak bottom too
       SBC HL,BC          }
       LD (msth),HL       }
       LD BC,12CØ         ; BC=12CØ
       LD HL,684Ø         ; HL=684Ø
       LD DE,175Ø         ; DE=175Ø
       PUSH DE            ; (dest_address) goto Reclaim in Home Rom
       LD DE,FFØØ         ; DE=FFØØ
       PUSH DE            ; (bank/chunk)
```

```
        LD DE,ØØØØ          ;DE=ØØØØ
        PUSH DE             ; no parms
        PUSH DE             ; no parms
        CALL CBNK(65DØ)     ;goto home rom and return
        JR ØF3D             ;jump forward
ØF3A -> SCF                 ;set carry flag
        JR ØF3E             ;jump forward
ØF3D -> AND A               ; reset carry flag
ØF3E -> POP AF              }
        POP HL              }
        POP DE              } Restore Regs
        POP BC              }
        RET                 ;and Return
ØF43 -> LD BC,(chad)        ;get character address
Passing ↑ CALL 255D         ;
        LD HL,(chad)        ;get character address
        AND A               ;clear carry
        SBC HL,BC           ;
        DEC HL              ;dec HL by one
        LD A,L              ;put L into A
        LD HL,(stnd)        ;get end of used memory
        LD (HL),A           ;put L there (A)
        INC HL              ;next address
        POP BC              ;get BC
        LD (HL),B           ;put B there
        INC HL              ;next address
        LD (HL),C           ;put C there
        INC HL              ;next address
        LD (stnd),HL        ;new end address
        LD HL,(chad)        ;get character address
        DEC HL              ;dec by one
        BIT Ø,A             ;is bit Ø set?
        JR Z,ØF73           ;no - jump forward
ØF67 -> DEC A               ;yes - dec a by one
        LD B,(HL)           ;get character
        DEC HL              ;next address
        DEC A               ;dec a by one
```

Page 19
Exrom Cont

```
ØF6B -> JP M,ØF7D        ;if minus (128-255), jump
        LD C,(HL)        ;get character
        DEC HL           ;next address
        PUSH BC          ;save BC
        JR ØF67          ;and redo
ØF73 -> LD B,2Ø          ;B=2Ø (space?)
        AND A            ;clear carry ~~(AND 2Ø and A)~~
        RET Z            ;return if Zero
        LD C,(HL)        ;get character
        DEC HL           ;next address
        DEC A            ;dec A by one (counter?)
        PUSH BC          ;save BC
        JR ØF67          ;and redo
ØF7D -> LD HL,(stnd)     ;get stack end (calulator)
        DEC HL           ;reduce by one
        LD A,(HL)        ;get character
        DEC HL           ;reduce by one
        LD (stnd),HL     ;update stnd
        LD H,(HL)        ;H=(stnd) character
        LD L,A           ;put A into L
        PUSH HL          ;save HL
ØF89 -> RET              ;and return

ØF8A -> PUSH AF          ;save AF
Jump    LD A,(vmod)      ;get video mode
Bank    AND A.           ;is it Zero?
        JR Z,ØF95        ;yes - jump -7
        POP AF           ;NO - get AF
        JP FD32          ;goto routine in Chunk 7
ØF95 -> POP AF           ;get AF
        JP GBAN(Ø572)    ;in chuk 3 - goto it now

ØF99 -> PUSH AF          ;save AF
CALL    LD A,(vmod)      ;get video mode
Bank    AND A            ;is it Zero?
```

```
        JR Z, ØFA4          ; yes - jump
        POP AF              ; NO - restor AF
        JP FD9Ø             ; in Chunk 7 - Jump ->
ØFA4 -> POP AF              ; get AF restore it
        JP CBAN (65DØ)      ; in chunk 3 - Jump ->
ØFA8 ->
 thru   } Blank (ØØ)
ØFFF ->

1ØØØ ->
 thru   } RAM-RES Code in EXROM
1623 ->   transtered to RAM at boot-up

1624 ->
 thru   } Blank (ØØ)
174E ->

174F -> ADD A,B             ; trash, data, junk

175Ø ->
 thru   } Blank
17FF ->

18ØØ ->
 thru   } unused (FF)
1BFF ->

1CØØ ->
 thru   } BLANK (ØØ)
1CFF ->

1DØØ ->
 thru   } Fixup Table for relocation of RAM-RES
1D7B ->   code at Chunk 3/7
1D7C ->
 thru   } Blank (ØØ)
1EDB ->
```

EXROM Cont

1EDC → thru 1FFF → } Jumptable (~~Not used in EXROM, I think!!!~~ Used by Function dispatcher for ROM calls)

End of EXROM

Note: Cassette Routines are not covered here. See page 14 on back for more details.

Note: This disassembly is not fully played through!!! some routines I have no idea what they do. I have an idea of course, but they do things I cannot simulate right now. I will continue to study the Timex system until I completely understand this changling system! Please excuse the spelling errors. This was done in 2 days and I did not pay any attention to spelling. Thanks

Dan Mabes

8/18/85

How to use RAM-RES Code:

CALL_BANK : do : (will call address and return in specified bank)

Push destination address first
Push (BANK/CHUNK) specification next
Push Parm-lob (#Pushes * 2) next ←to routine
Push Parm-hib (#Pushes * 2) next →from routine
CALL CALL_BANK (65DØ)
Exit: parms on stack

XFER_BYTES : do : (will copy bytes from one bank to another or the same bank)
(SOURCE BANK / DEST BANK)
Push (BANK/CHUNK) specification first
Push (source-address) next
Push (destination address) next
Push (length) next
Push (direction) next (1=forward, FF = backard)
CALL XFER_BYTES (6732) - Exit: none

GOTO_EXTENTION_ROM (simply goes to EXROM, NO return)

HL = Address in EXROM
CALL GOTO_EXROM (6815)
EXIT: None

Service Code — FUNCTION DESPATCHER

Ø	subr	address ØØ68
1	ldby	ØØKC
2	eds2	Ø184
3	eds	Ø18Ø
4	cent	ØIAB
5	load	Ø5CC
6	merse	Ø6E5
7	save	Ø851
8	Change video	ØEA3
9	Border	ØØE5

10 = FFFF
11
12 SAME AS 10
13

16 ENABANK = 6499
17 = 6571
18 = 65CF
19 = 6721
20 = ~~FFFF~~ FFFF
21
22 } SAME AS 20
23
24

Note: the above cannot be called using the Function dispatcher as IX is destroyed. They are marked as reserved.

Function dispatcher:

~~PUSH PARM-OUT (Pushes *2) first~~

set up for routine as if calling it directly on machine stack and calculator stack
~~We~~ Now:

Push PARM-OUT (PUSHES *2) first ← to routine
Push Parm-in (Pushes *2) next ← from routine
Push Service Code next (bit 15 set = Jump, 15 Reset = CALC)

Exit: Parms on stack (if any)

GET_WORD:

~~Address to~~ (gets a word in the designated bank at designated address)

HL = address
B = bank address is in
CALL 6316
EXIT = the word is in HL

PUT_WORD

puts a word into the specified bank

WORD to put in = DE
Address to put it in = HL
Bank to put it in = B
CALL 633B
No exit

WRITE_Bank_Regester

Regester Address in D
Regisger DATA in E
CALL 635C
no exit

(used for Expansion bank access, however, does not seem to really do much with real expansion banks)

READ_Bank_Regisger

Reads the Bank Switching Regisgers for Expansion Banks, however, does not seem to really do much with real expansion banks)

LSN_Address in D
MSN-Address in E
CALL 63AD

EXIT:

Byte_Data in = E

GET_BANK_STATUS

gets status for all banks (does it work with Exp Banks?)

Bank in = B
CALL 6445
Exit:

Bank status in = B
Chunk selection in = C

GET_CHUNK

get Chunk will get what chunk will need to be enabled for the address in HL.
Entry:
Address for chunk selection in = HL
CALL 644D
EXIT
The full chunk byte in = A

GET_Bank-Number

gets the bank number of the address in HL
CALL 645E
Address to check for bank in = HL

Exit:

Address in HL is in Bank = B

BANK_ENABLE

enables specified Bank and chunks

Bank to enable in = B
Chunks to enable in = C
CALL 6499
Exit:

None exit

SAVE_BANK_STATUS

pushes the status of all banks on the stack.

Enter - Nothing
CALL 651E
Exit
status_addresses in IX (for machine stack)

RESTORES_BANK_STATUS

restores bank statuses

Enter:

status_address in IX (for machine stack)
CALL 654A
Exit:

all bank restored

GOTO_BANK

goto bank in the specified bank — no return

Enter

Push dest_address first
Push (bank/chunk) next
CALL 6572
Exit:

None

MOVE_BYTES

This routine should not be called by itself as it uses the stack extensivly and is called from XFER_BYTES so use XFER_BYTES istead of this routine directly. However, this routine could be rewritten so it does not use the stack INDIRECTLY but uses it FOR ITS OWN USE. See MOVE_BYTES Modified SHEET. →

CREATE_BITMAP

This routine creates a ~~bit~~ chunk active for the address in HL. However, this routine should not be called directly as it uses the stack extensivly and should be called from XFER_BYTES only. This routine could be rewritten so it does not use the stack at all!! ~~to do this~~

~~Note~~

But WHY?? Just use the routine GET_CHUNK directly as all this routine does extra it handles the XFER_BYTES protocol for finding the chunks active for both source_bank and dest_bank and also determines the chunks if both source_bank and dest_bank are the same. In general, just use GET_CHUNK repeatly if necessary, to get desired result.

END OF RAM-RES routine descriptions

Jon Mc

6/18/85

XFER BYTES:

```
PUSH AF
PUSH BC
PUSH DE          } SAVE REGS
PUSH HL
LD HL,0000
ADD HL,SP        (HL now holds SP)
LD DE,000A
ADD HL,DE        (go back 10 bytes) (points to 5F74)
EX DE,HL
LD A,(6315)      ;get Bank_Switch_Maxbanks
LD C,A           ;put into C
LD B,00          ;clear B (BC holds Bank_Switch_Maxbanks)
LD HL,0000       ;clear HL
ADD HL,SP        ;[redacted] top of stack (HL holds this)
AND A            ;set flags (or reset in this case the carry flag)
SBC HL,BC        ;HL points to Max_banks - SP (5F6A)
DEC HL           ;↓
DEC HL           ;HL = SP - Max_banks - 2 (5F67)
PUSH HL          ;Transfer HL into...
POP IX           ;IX holds HL now
LD SP,IX         ;make stak pointer IX (5F67) (points to save_states)
CALL BSST(6SIE)  ;call save_status (of all banks)
PUSH DE          ;Save Parms address...
POP IX           ;and put into IX
LD L,(IX+06)     ;↓
LD H,(IX+07)     ;HL now holds src-addr (address which data is to be taken)
CALL BMAP(66E8)  ;call create_bitmap
PUSH AF          ;save [redacted] chunk of source bank temp.
LD L,(IX+04)     ;↓
LD H,(IX+05)     ;HL now holds dest-addr (address to which data is to be put)
CALL BMAP(66E8)  ;call create_bitmap
LD C,A           ;C = dest bit_map (chunk)
POP AF           ;get source bit_map
LD B,A           ;B = source bit_map (chunk)
LD A,(IX+09)     ;get source bank
LD D,(IX+08)     ;get dest bank
```

C=FB

Change to	CP D	;are we XFER_Bytes in the same bank?
	JR NZ,6760 (Diff_banks)	; No - goto Diff_banks
is a error	LD A,B	; Yes, banks match get source chunk
	AND C	; get a merge of chunks (turn on both chunks)
LD B,A →	LD E,A	; put into E (Note: This was an error!!!)
[redacted]	JR 6778 (do_move)	; and XFER those bytes
Diff banks	LD A,B	;get source chunk enable
	OR C	; are source and dest chunks same???
	CP FF	; check now
	JR NZ, Overlap (67A0)	; Yes, chunks same - jump to Overlap
	LD E,B	;get source chunk enable and put into 'E'
	LD B,D	;put dest-bank in 'b'
	CALL Enab (6499)	; Now enable dest-bank chunks
	LD B,(IX+09)	;get source-bank
	LD C,E	; put chunk status into 'C'
	CALL ENAB (6499)	; Now enable source-bank chunks
	LD L,(IX+06)	;↓
	LD H,(IX+07)	;get source address
	LD E,(IX+04)	;↓
	LD D,(IX+05	;get destination address
	LD C,(IX+02)	;↓
	LD B,(IX+03)	; get length
	LD A,(IX+00)	; get direction
	RLCA	; If FF (backward direction), put into carry flag
	RRCA	; put back to normal
	JR C, Reverse (67AC)	; If carry set (FF direction), goto Reverse
	LDIR	; No - go foward direction. now copy.
	JR EXIT (67EE)	; and EXIT
Reverse	LDDR	; backward direction. now copy.
	JR EXIT (67EE)	; and EXIT
Overlap	LD HL, MSTBOT	; HL = [redacted] → address of this variable
	PUSH BC	;save chunk status
	LD B,255	; B = Home bank (FF)
	CALL get-word	; get word at (HL) (5CC0)
	POP BC	
	LD DE,STKSZ	
	AND A	

```
            XFER_BYTES
            SBC HL,DE
            LD DE,Free_Bytes
            ADD HL,DE
            EX DE,HL
            LD HL,0000
            ADD HL,SP
            INC DE
            AND A
            SBC HL,DE
            JR NC,XB-Space
            LD A,01
            JR Exit
XB_Space    DEC DE
            EX DE,HL
            LD SP,HL
            INC DE
            LD A,(IX+Direction) (IX+00)   ;get direction
            LD (IX+  ),L
            LD (IX+  ),H
            ~~LD (IX+02),~~
            LD L,(IX+03)                  ;z
            LD H,(IX+04)                  ;get length
move_loop   AND A
            SBC HL,DE
            JR C,last_Move
            CALL Move_bytes
            JR move_loop
last_move   ADD HL,DE
            EX DE,HL
            CALL move_bytes
            EX DE,HL
            LD L,(IX+  )
            LD H,(IX+  )
            ADD HL,DE
            LD SP,HL
```

```
Exit    XOR A               ;Clear A Reg
        LD IX,0000          ;clear IX Reg
        ADD IX,SP           ;IX holds Stack pointer to bank status in stack
        CALL RBST (654A)    ;Restore all banks to normal
        INC IX              ;IX holds Stack address
        LD SP,IX            ;put IX into SP (restore stack)
        POP HL              ;get old HL
        POP DE              ;get old DE
        POP BC              ;get old BC
        POP AF              ;get old AF
        POP IX              ;get address of Calling Routine<-
        EX (SP),IX          ;put IX into stack (cleans up direction flag)
        POP IX              ;get address of Calling Routine<-
        EX (SP),IX          ;put IX into stack (cleans up length)
        POP IX              ;get address of calling Routine
        EX (SP),IX          ;Put IX into stack (cleans up dest-addr)
        POP IX              ;get address of calling Routine
        EX (SP),IX          ;Put IX into stack (cleans up source-addr)
        POP IX              ;get address of calling Routine
        EX (SP),IX          ;Put IX into stack (cleans up bank selects)
        RET                 ;Return to calling Routine
```

Write Bank-switch Register

```
PUSH AF        }
PUSH BC        } SAVE REGS
PUSH HL        }
LD A,D         ; Put reg-address in 'H'
LD L,00        ; L=0 so HL = XX00 address (memory mapped)
LD A,(C000)    ; LD A with the contents of C000
PUSH AF        ; save contents
LD A,(HL)      ; Get contents of XX00 address (lownyb)
PUSH AF        ; Save contents
LD A,07        ; A = 7 (Sound Enable Reg)
OUT (F5),A     ; set it
IN A,(F6)      ; get contents of Reg 7
LD B,A         ; Put into B
LD A,0E        ; A=14 (I/O Port Reg)
[illegible]OUT (F5),A  ; set it
IN A,(F6)      ; Pull in I/O Port A
LD C,A         ; Put into C
LD A,07        ; A=7 (Enable Reg)
OUT (F5),A     ; set it
LD A,40        ; A=40 (64 decimal)
OUT (F6),A     ; set Enable for the I/O Port
LD A,0E        ; A=0E (14 decima) (I/O Port Reg)
OUT (F5),A     ; Set it
XOR A          ; Clear A Reg
OUT (F6),A     ; Send a 0 out the port
LD A,02        ; A=2
LD (C000),A    ; Put A in C000 (Reset nibble steering logic) ?
LD A,E         ; get reg-data (least sig. nybble)
LD (HL),A      ; put into XX00 address (lownyb)
SRA A          }
SRA A          } get most significant nybble
SRA A          }
SRA A          }
LD (HL),A      ; put into XX00 address (lownyb)
LD A,07        ; Enable Reg on Sound chip
```

```
OUT (F5),A      ;set it
LD A,B          ;get old value
OUT (F6),A      ;and restore it
LD A,0E         ;Get the I/O Port Reg
OUT (F5),A      ;set it
LD A,C          ;set old value
OUT (F6),A      ;and restore it
POP AF          ;get old contents of XX00 (low nyb)
LD (HL),A       ;and restore it
POP AF          ;set old contents of C000 (low nyb)
LD (C000),A     ;and restore it
POP HL          }
POP BC          } Restore Regs
POP AF          }
RET             ;Return to calling routine
```

1) GOTO_BANK

ENTER: PUSH TARGET ADDRESS ON STACK (HL)
· PUSH BANK #/MEMORY CHUNK SELECT ON STACK (BC)

EXIT: NO EXIT FROM THIS ROUTINE

EXAMPLE 1:

JUMPING TO THE TAPE ROUTINE MASTER CONTROL PROGRAM:

```
LD HL,0068h (Tape routine)
PUSH HL
LD BC,FEFEh (FE=BANK (EXTENTION); FE=CHUNK SELECT)
PUSH BC
CALL ~~JMP~~ GOTO_BANK (6572h)
```

JP does not work.

Note: This example will cause a non-recoverable crash.
Use only for demonstration and learning example.

Note: Must correct "BANK-ENABLE" errors before using.

NOTE: MAY NEED TO REVERSE THE STACK OPERATIONS

Dortha Wellbanks

Mary Wellbanks 4-Seasons [5423] Juan tubo
Knocked off antenna

[293-3756]

SAVE STATUS

CREATE BITMAP

TI 2068 RAM-RES CODE

GOTO - EXTENSION ROM

ENTRY: MUST BE CALLED WITH ADDRESS TO GOTO IN EXROM
IN THE 'HL' REGISTER PAIR

EXIT: NO RETURN FROM THIS ROUTINE UNLESS YOU PUSH
A RETURN ADDRESS ON THE STACK FOR RETURN

SOURCE CODE: ENTRY: 6815 (1615 IN EXROM)

```
POP IX              ; Trash return address
PUSH AF             ; Save 'AF'
IN A,(FF)           ; Read EXROM Port
SET 7,A             ; Set EXROM 'ON'
OUT (FF),A          ; ENFORCE IT NOW
LD A,01             ; LOAD A WITH 1
OUT (F4),A          ; SET SELECT REGISTER
POP AF              ; Restore 'AF'...
JP (HL)             ; And jump to (HL)
```

```
936B 00          NOP
936C 00          NOP
936D 00          NOP
936E 00          NOP
936F 00          NOP
9370 00          NOP
9371 00          NOP
9372 011800      LD BC,0018
9375 119C3A      LD DE,3A9C
9378 CDA400      CALL 00A4
937B 016600      LD BC,0066
937E 110080      LD DE,8000
9381 21E913      LD HL,13E9
9384 EDB0        LDIR
9386 C30080      JP 8000
9389 303D        JR NC,93C8
938B C33480      JP 8034
938E 0604        LD B,04
9390 3E10        LD A,10
9392 57          LD D,A
9393 D3F4        OUT (F4),A
9395 DBF4        IN A,(F4)
9397 BA          CP D
9398 2048        JR NZ,93E2
939A CB27        SLA A
939C 10F4        DJNZ,9392
939E 3EFF        LD A,FF
93A0 57          LD D,A
93A1 D3F4        OUT (F4),A
93A3 DBF4        IN A,(F4)
93A5 BA          CP D
93A6 203A        JR NZ,93E2
93A8 D9          EXX
93A9 79          LD A,C
93AA D9          EXX
93AB D3F4        OUT (F4),A
93AD 012600      LD BC,0026
93B0 11B43A      LD DE,3AB4
93B3 CDB000      CALL 00B0
93B6 CDE800      CALL 00E8
93B9 CDBD00      CALL 00BD
93BC CDD300      CALL 00D3
93BF CDE300      CALL 00E3
93C2 C25914      JP NZ,1459
93C5 C34800      JP 0048
93C8 011C00      LD BC,001C
93CB 11B43A      LD DE,3AB4
93CE CDB000      CALL 00B0
93D1 011B00      LD BC,001B
93D4 11DA3A      LD DE,3ADA
93D7 CDB000      CALL 00B0
93DA CDE800      CALL 00E8
93DD CDCB00      CALL 00CB
93E0 18DA        JR 93BC
93E2 D9          EXX
93E3 79          LD A,C
93E4 D9          EXX
93E5 D3F4        OUT (F4),A
93E7 18DF        JR 93C8
93E9 F3          DI
93EA DBFF        IN A,(FF)
93EC 4F          LD C,A
93ED AF          XOR A
93EE 0608        LD B,08
93F0 57          LD D,A
93F1 D3FF        OUT (FF),A
93F3 DBFF        IN A,(FF)
93F5 BA          CP D
93F6 201E        JR NZ,9416
93F8 3C          INC A
93F9 57          LD D,A
93FA D3FF        OUT (FF),A
93FC DBFF        IN A,(FF)
93FE BA          CP D
93FF 2015        JR NZ,9416
9401 CB27        SLA A
9403 10F4        DJNZ,93F9
9405 3EFF        LD A,FF
9407 57          LD D,A
9408 D3FF        OUT (FF),A
940A DBFF        IN A,(FF)
940C BA          CP D
940D 2007        JR NZ,9416
940F 79          LD A,C
9410 D3FF        OUT (FF),A
9412 37          SCF
9413 C38913      JP 1389
9416 79          LD A,C
9417 D3FF        OUT (FF),A
9419 A7          AND A
9419 C38913      JP 1389
```

93D2

Change to '3'

Error

93EC

Cannot Do an 'FF'

Use '73'

93E (Chunks 0-1, 4-7)

USE '73'

(Chunks 0-1)

(Chunks 4-6)

- Normal!

```
93DA CDE800          CALL 00E8
93DD CDCB00          CALL 00CB
93E0 18DA            JR 93BC
93E2 D9              EXX
93E3 79              LD A,C
93E4 D9              EXX
93E5 D3F4            OUT (F4),A
93E7 18DF            JR 93C8
93E9 F3              DI
93EA DBFF            IN A,(FF)
93EC 4F              LD C,A
93ED AF              XOR A
93EE 0608            LD B,08
93F0 57              LD D,A
93F1 D3FF            OUT (FF),A
93F3 DBFF            IN A,(FF)
93F5 BA              CP D
93F6 201E            JR NZ,9416
93F8 3C              INC A
93F9 57              LD D,A
93FA D3FF            OUT (FF),A
93FC DBFF            IN A,(FF)
93FE BA              CP D
93FF 2015            JR NZ,9416
9401 CB27            SLA A
9403 10F4            DJNZ,93F9
9405 3EFF            LD A,FF
9407 57              LD D,A
9408 D3FF            OUT (FF),A
940A DBFF            IN A,(FF)
940C BA              CP D
940D 2007            JR NZ,9416
940F 79              LD A,C
9410 D3FF            OUT (FF),A
9412 37              SCF
9413 C38913          JP 1389
9416 79              LD A,C
9417 D3FF            OUT (FF),A
9419 A7              AND A
941A C38913          JP 1389
941D F3              DI
941E DBF4            IN A,(F4)
9420 D9              EXX
9421 4F              LD C,A
9422 D9              EXX
9423 AF              XOR A
9424 57              LD D,A
9425 D3F4            OUT (F4),A
9427 DBF4            IN A,(F4)
9429 BA              CP D
942A 201B            JR NZ,9447
942C 3C              INC A
942D 57              LD D,A
942E D3F4            OUT (F4),A
9430 DBF4            IN A,(F4)
9432 BA              CP D
9433 2012            JR NZ,9447
9435 CB27            SLA A
9437 57              LD D,A
9438 D3F4            OUT (F4),A
943A DBF4            IN A,(F4)
943C BA              CP D
943D 2008            JR NZ,9447
943F D9              EXX
9440 79              LD A,C
9441 D9              EXX
9442 D3F4            OUT (F4),A
9444 C38E13          JP 138E
9447 D9              EXX
9448 79              LD A,C
9449 D9              EXX
944A D3F4            OUT (F4),A
944C C3C813          JP 13C8
944F FF              RST 38H
9450 FF              RST 38H
9451 FF              RST 38H
9452 FF              RST 38H
9453 FF              RST 38H
9454 FF              RST 38H
9455 FF              RST 38H
9456 FF              RST 38H
9457 FF              RST 38H
9458 FF              RST 38H
```

-Normal!

Normal=3

(FF)=0 -Normal

(FF)=0 -Chunk 0

(FF)=0 -Chunk 1

-(F4=3)

}(F4)=3

13E9= 93F1

```
Header Code-12/01/85-C. Miles

8000 FF          255 32768 COPY
8001 01            1 32769
8002 13           19 32770
8003 00            0 32771
8004 FC          252 32772 DRAW
8005 FF          255 32773 COPY
8006 FF          255 32774 COPY
8007 FF          255 32775 COPY

8008 C36418          JP 1864
800B FF              RST 38H
800C FF              RST 38H
800D FF              RST 38H
800E FF              RST 38H
800F FF              RST 38H
8010 C3F215          JP 15F2
8013 F3              DI
8014 2AB25C          LD HL,(rmtp)
8017 363E            LD (HL),3E
8019 2B              DEC HL
801A F9              LD SP,HL
801B 21B65C          LD HL,err\
801E 224F5C          LD (chan),HL
8021 11AF15          LD DE,15AF
8024 1805            JR 802B
8026 FF              RST 38H
8027 FF              RST 38H
8028 C35B33          JP 335B
802B 011500          LD BC,0015
802E EB              EX DE,HL
802F EDB0            LDIR
8031 21C615          LD HL,15C6
8034 1804            JR 803A
8036 FF              RST 38H
8037 FF              RST 38H
8038 FB              EI
8039 C9              RET
803A 11105C          LD DE,strm
803D 010E00          LD BC,000E
8040 EDB0            LDIR
8042 21925C          LD HL,memb
8045 22685C          LD (mem_),HL

8048 3E03            LD A,03
804A CD282D          CALL 2D28
804D CD9422          CALL 2294
8050 3A8D5C          LD A,(atrp)
8053 0F              RRCA
8054 0F              RRCA
8055 0F              RRCA
8056 E6F8            AND F8
8058 C605            ADD A,05
805A 07              RLCA
805B 07              RLCA
805C 07              RLCA
805D 328D5C          LD (atrp),A
8060 3EF8            LD A,F8
8062 FDA653          AND (atrp)
8065 328D5C          LD (atrp),A
8068 01BA01          LD BC,01BA
806B 116E38          LD DE,386E
806E CDA400          CALL 00A4

8071 01FEF7          LD BC,F7FE
8074 ED78            IN A,(C)
8076 FE1E            CP 1E
8078 2878            JR Z,80F2
807A FE1D            CP 1D
807C CA4802          JP Z,0248
807F FE1B            CP 1B
8081 CA6B13          JP Z,136B
8084 FE17            CP 17
8086 CA5914          JP Z,1459
8089 FE0F            CP 0F
808B CAF614          JP Z,14F6
808E 01FEEF          LD BC,EFFE
8091 ED78            IN A,(C)
8093 FE0F            CP 0F
8095 CACE17          JP Z,17CE
8098 FE17            CP 17
809A CA6C18          JP Z,186C
809D FE1B            CP 1B
809F CADA19          JP Z,19DA
80A2 18CD            JR 8071

80A4 C5              PUSH BC
80A5 D5              PUSH DE
```

```
804A CD282D         CALL 2D28
804D CD9422         CALL 2294
8050 3A8D5C         LD A,(atrp)
8053 0F             RRCA
8054 0F             RRCA
8055 0F             RRCA
8056 E6F8           AND F8
8058 C605           ADD A,05
805A 07             RLCA
805B 07             RLCA
805C 07             RLCA
805D 328D5C         LD (atrp),A
8060 3EF8           LD A,F8
8062 FDA653         AND (atrp)
8065 328D5C         LD (atrp),A
8068 01BA01         LD BC,01BA
806B 116E38         LD DE,386E
806E CDA400         CALL 00A4

8071 01FEF7         LD BC,F7FE
8074 ED78           IN A,(C)
8076 FE1E           CP 1E
8078 2878           JR Z,80F2
807A FE1D           CP 1D
807C CA4802         JP Z,0248
807F FE1B           CP 1B
8081 CA6B13         JP Z,136B
8084 FE17           CP 17
8086 CA5914         JP Z,1459
8089 FE0F           CP 0F
808B CAF614         JP Z,14F6
808E 01FEEF         LD BC,EFFE
8091 ED78           IN A,(C)
8093 FE0F           CP 0F
8095 CACE17         JP Z,17CE
8098 FE17           CP 17
809A CA6C18         JP Z,186C
809D FE1B           CP 1B
809F CADA19         JP Z,19DA
80A2 18CD           JR 8071

80A4 C5             PUSH BC
80A5 D5             PUSH DE
80A6 3E02           LD A,02
80A8 CD0116         CALL 1601
80AB CD6B0D         CALL 0D6B
80AE 1802           JR 80B2
80B0 C5             PUSH BC
80B1 D5             PUSH DE
80B2 3E02           LD A,02
80B4 CD0116         CALL 1601
80B7 D1             POP DE
80B8 C1             POP BC
80B9 CD3C20         CALL 203C
80BC C9             RET

80BD 3E02           LD A,02
80BF CD282D         CALL 2D28
80C2 3E1E           LD A,1E
80C4 CD282D         CALL 2D28
80C7 CDF803         CALL 03F8
80CA C9             RET

80CB 3E02           LD A,02
80CD CD282D         CALL,2D28
80D0 AF             XOR A
80D1 18F1           JR 80C4

80D3 FB             EI
80D4 063C           LD B,3C
80D6 76             HALT
80D7 10FD           DJNZ,80D6
80D9 F3             DI
80DA CD8E02         CALL 028E
80DD 1C             INC E
80DE AF             XOR A
80DF BB             CP E
80E0 28F8           JR Z,80DA
80E2 C9             RET

80E3 3AB05C         LD A,(5CB0)
80E6 A7             AND A
80E7 C9             RET

80E8 011E00         LD BC,001E
80EB 117715         LD DE,1577
80EE CDB000         CALL 00B0
80F1 C9             RET
```

EXTRA CIRCUIT ON TS2008

(CONTROLS CHUNKS 2&3) "UPSIDE DOWN BOARD"

	$\overline{BE}$	$\overline{ROSCS}$	$\overline{WR}$	
Normal #1	1	1	1	→ SCLD $\overline{TS}$ TAKES CONTROL OF RAM
Normal #2	1	1	Ø	→ " " " " " "
CARTRIDGE #1	1	Ø	1	→ " " " " " "
CART #2	1	Ø	Ø	→ " " " " " "
EXTERNAL #1	Ø	1	1	→ " " " " " "
" #2	Ø	1	Ø	→ " " " " " "
" #3	Ø	Ø	1	→ " " " " " "
" #4	Ø	Ø	Ø	→ " " " " " "

In normal TS2068 operation, the upside down board does nothing. But, when Bank Enable is held low (indicating external memory is to be accessed and internal memory to be shut off)

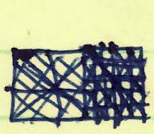
NAND 00=1 01=0 10=0 11=0

AND 00=0 01=0 10=0 11=1

	$\overline{BE}$	$\overline{ROSCS}$	$\overline{WE}$	Heat	
Normal #1	1	1	1	1	$\overset{0}{0} > 1 - \overset{1}{1} > \phi$
Normal #2	1	1	ϕ	1	$\overset{0}{0} > 1 - \overset{1}{z} > \phi$
CART #1	1	ϕ	1	1	$\overset{0}{0} > 1 - \overset{1}{1} > 0$
CART #2	1	ϕ	ϕ	1	$\overset{0}{0} > 1 - \overset{1}{1} > \phi$
EXTN #1	ϕ	1	1	1	$\overset{0}{0} > 1 - \overset{1}{1} > \phi$
EXTN #2	ϕ	1	ϕ	1	$\overset{0}{0} > 1 - \overset{1}{1} > \phi$
EXTN #3	ϕ	ϕ	1	1	~~$\overset{1}{\phi} > 1 - \overset{1}{1} > \phi$~~ $\overset{1}{\phi} > \phi, \overset{1}{\phi} > \phi$
EXTN #4	ϕ	ϕ	ϕ	1	$\overset{1}{0} > 0 - \overset{1}{\phi} > \phi$

Ray 988-2096

Low Power Supply
High Power Supply
Burned Transistor
Double Video/Striped Video (SCLD)
No Load/Save (Descrete Components)
RAM (low $\overline{RFSH}$ line) (Overwrites RAM) (IN BASIC)
Decoders (For RAM)
Buffers (for BUSS)
ROMS



Tim Carver (513) 751-8277
TS Connection -
Tommorow At 4-4:30 PM (2-2:30 Our Time)

11110001
00001110
00001100

$\boxed{\phi D}$

$\boxed{7\phi 81}$
$\boxed{7\phi BC}$

GB
BB
QE
PP

TAPES

OR

00 = 0
01 = 1
10 = 1
11 = 1

WAND

00 = 1
01 = 1
10 = 0
11 = 0

AND

00 = 0
01 = 0
10 = 0
11 = 1

Boot Hz EXROM with USR 32768

Carl

NEW COMMANDS FOR HOT Z AROS V 2.45

A few new commands have been added for creating, viewing and printing ASCII files.

Write Mode

POINT Creates an ASCII source file that could be edited and used with an assembler. The code in whatever bank is active is disassembled, the address and hexcode columns are discarded, and the lines are terminated with a semicolon. The ASCII codes are sent to a file in HOME bank at the address determined by the pointer DES2 (5FD8). You must set that pointer manually by writing directly to it with HOT Z. At the end of the operation DES2 will point to the end of the file, so you could use this command successively to create one file from several separate segments of memory. The disassembly begins with the cursor address and finishes at END, which must be set in advance. You must have assigned a label to every jump or call address if you expect the file to be palatable to an assembler.

CODE Creates an ASCII file of everything on the HOT Z screen from the cursor address to END. The file is created at the address contained in DES2 (5FD8) in HOME RAM. At the end of this operation, DES2 will point to the end of ASCII file. Move that address to END with the TO command for printing, saving, or viewing the file. You should be able to get at such files with a word processor in order to add annotations for archiving. Be sure to set DES2 low enough that your file will fit below FFFF, as there is no check for overflow.

SCREEN$ Prints an ASCII file to screen. Set the cursor to the first byte of the file and set END where you want to stop. Printing will pause for the Sinclair "scroll?" after 22 lines, and you can break with the space bar or continue by hitting ENTER. This command is for viewing only; it does not allow you to edit the file.

LPRINT Prints an ASCII file to a line printer. Prints from the cursor address to END. You can interrupt with the BREAK key.

Cat 1

There is also a new command for comparing blocks of memory across banks.

CIRCLE Set the cursor to the first address of one block and set END to the first address of the block to be compared. (This could be the same address if the blocks are in different banks.) When you give the command, you will be asked for source and destination banks: enter the banks of the two memory blocks. The comparison will begin at once and the display will jump to the first address where the memory contents differ. It important that you only use this command with the cursor set at an EVEN (0,2,4,6,8,A,C,E) numbered address. The purpose is to find small differences in two blocks of code; the command will not be useful for blocks that differ greatly.

One new READ mode command for moving the display:

NOT Moves the display to the address in END.

And one for moving HOT Z:

VAL$ Moves HOT Z to the EXROM bank, at the same addresses, if you have modified your 2068 to have memory there, and turns control over to the new version.

Title Page

TIS 2068

(TIMEX/SINCLAIR 2068 COLOR COMPUTER)

DIAGNOSTICS CARTRIDGE

© 1985 By: Carl A. Mees

VERSION ART 1.0 - Nov 10, 1985

OPERATION MANUAL

VERSION ART 1.0 - Nov 11, 1985

DIAGNOSTICS CARTRIDGE VERSION ART 1.0

BY: CARL A. MILES
© 1985

This is an LROS Cartridge designed to test the Timex/Sinclair 2068 Computer for defects in key hardware and software functions. In most cases, a simple PASS/FAIL result is given. However, in some cases, furthur testing will be needed on the part of the technician testing and repairing the computer. The tests are as follows:

1) Home RAM Test
2) CPU Test
3) SCLD Test
4) Home ROM Test
5) Extension Rom Test
6) Cassette Test
7) Keyboard Test
8) Test All Functions

The above test appear in the Main Menu exactly as shown above. I will explain how each test is to be used and how the result of each test should be interpeted.

Some tests use the Home RAM as a temporary storage area for running programs ~~and extended ting checksums~~ and therefore will NOT function if the Home RAM Test has ~~FAILED~~ or ~~the Home~~ has not yet been run. These tests are as follows:

1) SCLD Test
2) Home ROM Test
3) Extension Rom Test

~~2) Some other tests~~ The Cassette Test uses very little Home RAM

and ~~are~~ can function without the Home RAM test's ~~being run~~ permission. ~~These tests are:~~

~~1) Cass~~

The CPU Test and Keyboard Test do NOT use any Home RAM ~~to~~ to function. Therefore, these tests can be run regardless of if the Home RAM is good or bad.

THE TESTS

1) Home RAM Test

The Home RAM Test will test all 48K of the Home RAM. ~~It~~ This test is run ten times for the sake of complete safety. If the Home RAM can pass ten tests, then odds are that the Home RAM is indeed good. If this test finds an error, it will report it and continue to the next 16K block of memory (the TTS 2068's Home RAM is split into 3 blocks of 16K each) and continue the test. If an error is found in the last 16K block, it will report it, ~~and~~ lock the Home RAM dependent programs from use, and end. Check the chips given by the test in the order shown. The ~~chips~~ RAM chips should be COOL to the touch. If ~~they~~ ~~the~~ a RAM chip~~s~~ ~~are~~ is HOT to the touch, the RAM chip is BAD. Replace it. The other chips must be tested. The quickest method is ~~the~~ to take the suspected part, put it into a new, good machine, and see if the new machine works correctly. If ~~it~~ the new machine DOES work, the part is OK; however, if the new machine fails to work, the part is bad. Replace it.

The Home RAM Test is very strightforward and does not require any furthur information.

2) CPU Test

The CPU Test will put the Z-80A Microprocessor through its paces. Simply put, if this test passes, the CPU is OK; however, if this test fails, replace the CPU chip. Very strightforward.

3) SCLD Test

The SCLD Test will test the SCLD's bankswithing and system control ports. As with the CPU Test, if this test passes, the SCLD can bank-switch and control the computer correctly. However, this is not an idication that the SCLD is 100% OK. The Home RAM Test, this test, the Cassette test and the Keyboard test must all pass for the SCLD to be CONSIDERED good. The SCLD is a very complex chip that also controls the video memory and screen. If the video output is OK, then the SCLD can be CONSIDERED good. Please note that there is no way you can say the SCLD is 100% good. If it passes all the tests and the video is good, the SCLD can be considered 100% good, but don't count on it.

If the SCLD test fails, then replace the SCLD chip. Very strightforward.

4) Home ROM Test

The Home ROM Test is also very simple. If the test passes, the Home ROM is good. If the test failes, replach the Home ROM chip.

5) Extension Rom Test

As with the Home ROM Test, if this test passes, the EXROM is good. If the test fails, replace the EXROM chip. Very simple.

6) Cassette Test

This test will SAVE a 1K block of ROM to the tape five times. Set up tape for a SAVE and follow directions on the screen. After the test saves five time, the test will then verify the SAVES. Set up tape for loading and follow directions on the screen. If this test passes, all descrete components and the SCLD can be considered good. If this test fails, check all applicale descrete compombonents and, if necessary, the SCLD input and output pins for the cassette and buzzer.

7) Keyboard Test

This test will test all 42 keys including the spacebar. It also tests the SHIFTed function keys. The order of the keys need to be pressed are as follows:

1, 2, 3, 4, 5, 6, 7, 8, 9, 0
Q, W, E, R, T, Y, U, I, O, P
A, S, D, F, G, H, J, K, L, ENTER
CAPS SHIFT, Z, X, C, V, B, N, M, SYMBOL SHIFT, BREAK, CAPS SHIFT
SPACEBAR
CAPS SHIFT/5 (Cursor Left)
CAPS SHIFT/6 (Cursor Down)
CAPS SHIFT/7 (Cursor Up)
CAPS SHIFT/8 (Cursor Right)
SYMBOL SHIFT/M or '.' (decimal point, period)
SYMBOL SHIFT/V or '/' (backslash, divide symbol)
SYMBOL SHIFT/J or '-' (hyphen, subtraction symbol)

If you pressed all keys in the order above, the test will pass and will tell you so. All keys will produce a short 'beep' when accepted. If a key does not produce a 'beep', it could by you did not follow the sequence correctly

or the key is bad. If you followed the sequence incorrectly, press CAPS SHIFT/BREAK. The test will say the keyboard test failed. In fact, if you ever break out of the keyboard test this way, the test will indeed fail as YOU had to break out. Normally this does not occur. After the 'Press any key to continue...' messge, just rerun the keyboard test and follow the sequence to the letter.

If you have followed the sequence to the letter and a key still refuses to be accepted, the key has failed. Use the CAPS SHIFT/BREAK method described above to break out. If this is the case, the keyboard test has indeed FAILED. Check the physical keyboard itself and the diodes CR6 thru CR13. If these check out OK, then the SCLD chip is at fault. Replace the SCLD chip if this is the case. If the diodes are bad, replace the faulty ones and retest until the keyboard passes. If the physical keyboard itself is bad, either repair the keyboard or replace it. Then retest again.

~~Unless the SCLD chip itself has been determined to be the component at fault, the keyboard circuitry and/or the keyboard itself CAN be repaired and~~

8) Test All Functions

This is not itself a test. It simply sets up the program so the tests are performed in 'daisy chin' or one after the other. This is simply a conveince feature and should be used initialy, with the individal tests only called after the faulty have been identified and repaired.

hardware/ software

~~The feature will start with the Home Ram test and procede with the next test regardless of weither it fails the preives test passed or failed. It~~ is your responsibility to write down the failed tests and the chips to check and/or replace.

~~After running this feature, turn off the computer~~ and turn it on again.

When this feature is enabled and if it goes through ALL seven tests with all seven tests PASSING, the T/S 2068 under test can be considered good and should be put back together and retested one final time useing this feature. If it passes all seven test again, put that T/S 2068 into use.

IMPORTANT: Do not forget to record the serial number on the case of the T/S 2068 and the serial number on the actual P.C. Board. Also record what was wrong with the computer under test before it was fixed so if the same computer comes back, you can see what was wrong and what was done to repair it.

~~Diagnostics~~

T/S 2068 Diagnostics Cartridge

© 1985 By: Carl A. Miles

Version ART 1.0 - Nov 10, 1985

(4K) A12 0
(8K) A13 0
(16K) A14 0
(32K) A15 0
↓

A13	A14	A15	
0	0	0	= Ø-8K
1	0	0	= 8K-16K
0	1	0	= 16K-24K
1	1	0	= 24K-32K

A13	A14	A15	
0	0	1	= 32K-40K
1	0	1	= 40K-48K
0	1	1	= 48K-56K
1	1	1	= 56K-64K

~~Resistors~~ 6 # 10000 Mat. Resistors

U6 = SAME

U7 = SAME

U1 = SAME

U3 =

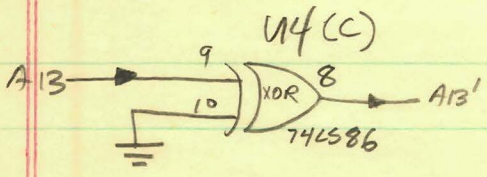
XOR

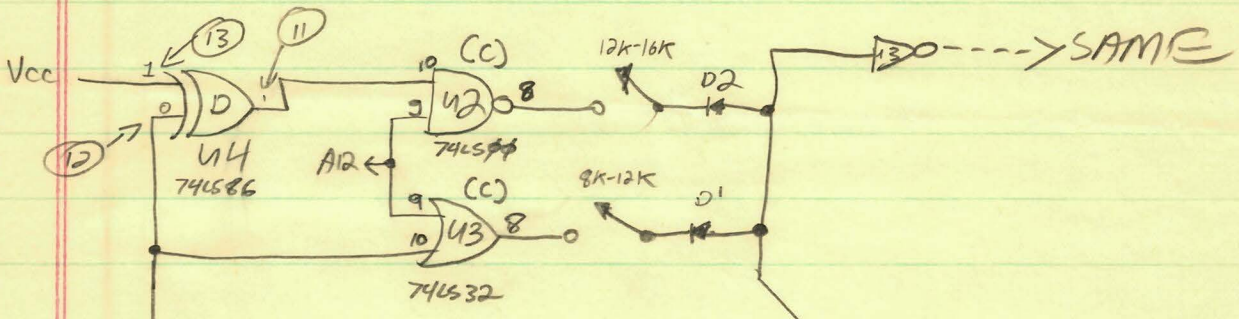
0 0 = 0

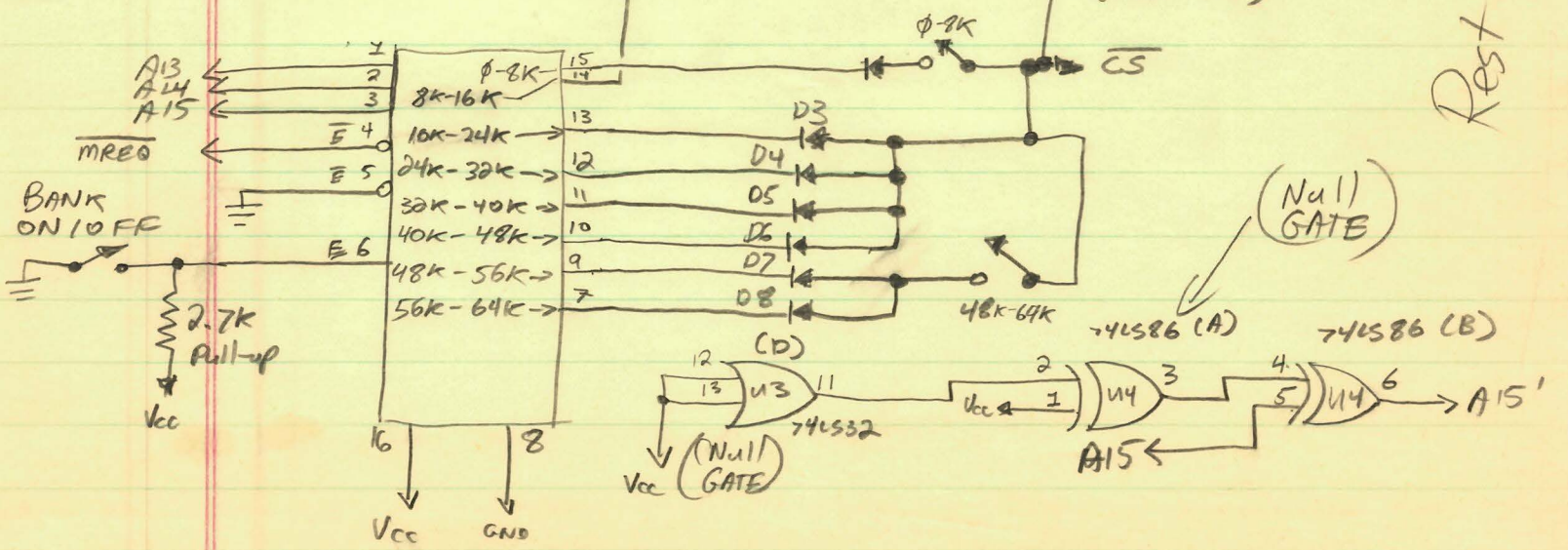
0 1 = 1

1 0 = 1

1 1 = 0



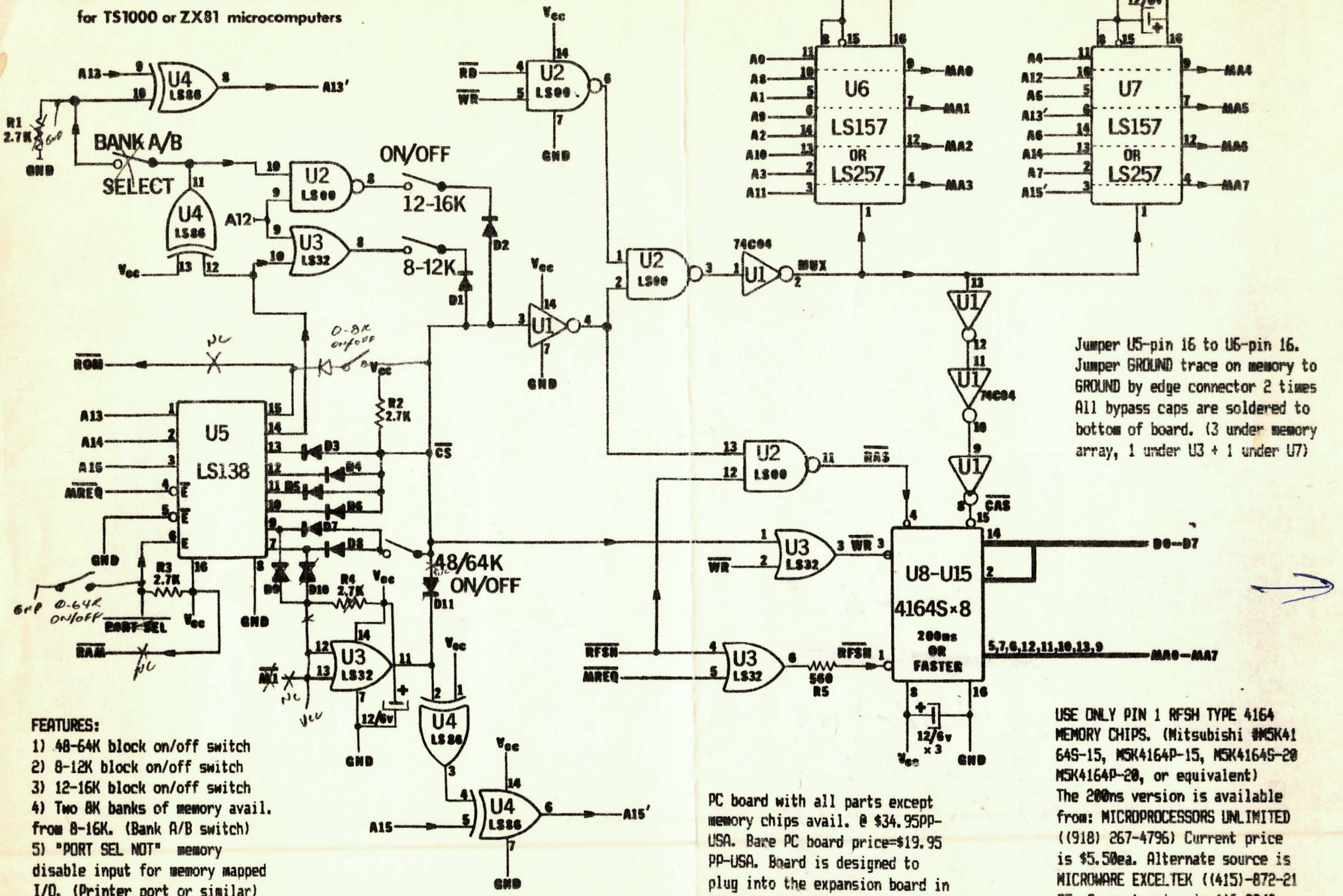




Rest of Circuit OK

64K BYTE DYNAMIC MEMORY

for TS1000 or ZX81 microcomputers



Jumper U5-pin 16 to U6-pin 16. Jumper GROUND trace on memory to GROUND by edge connector 2 times All bypass caps are soldered to bottom of board. (3 under memory array, 1 under U3 + 1 under U7)

FEATURES:

1) 48-64K block on/off switch
2) 8-12K block on/off switch
3) 12-16K block on/off switch
4) Two 8K banks of memory avail. from 8-16K. (Bank A/B switch)
5) "PORT SEL NOT" memory disable input for memory mapped I/O. (Printer port or similar)
6) Uses "PIN 1 RFSH" 4164 chips

John L. Oliger
10115 Nassau Lane
Indpls IN 46229

PC board with all parts except memory chips avail. @ $34.95PP-USA. Bare PC board price=$19.95 PP-USA. Board is designed to plug into the expansion board in Winter's SQ magazine. Board measures aprox. 5 1/2" X 3 1/4" Board material is G-10 or FR-4 epoxy with solder plated traces.

USE ONLY PIN 1 RFSH TYPE 4164 MEMORY CHIPS. (Mitsubishi #M5K4164S-15, M5K4164P-15, M5K4164S-20 M5K4164P-20, or equivalent) The 200ns version is available from: MICROPROCESSORS UNLIMITED ((918) 267-4796) Current price is $5.50ea. Alternate source is MICROWARE EXCELTEK ((415)-872-2195. Current price is $46.00/8.

JLO 8252

U1 - 74C04

U2 - 74LS00

U3 - 74LS32

U4 - 74LS36

U5 - 74LS86

U6 - 74LS157 or 257

~~U7 - 74LS~~

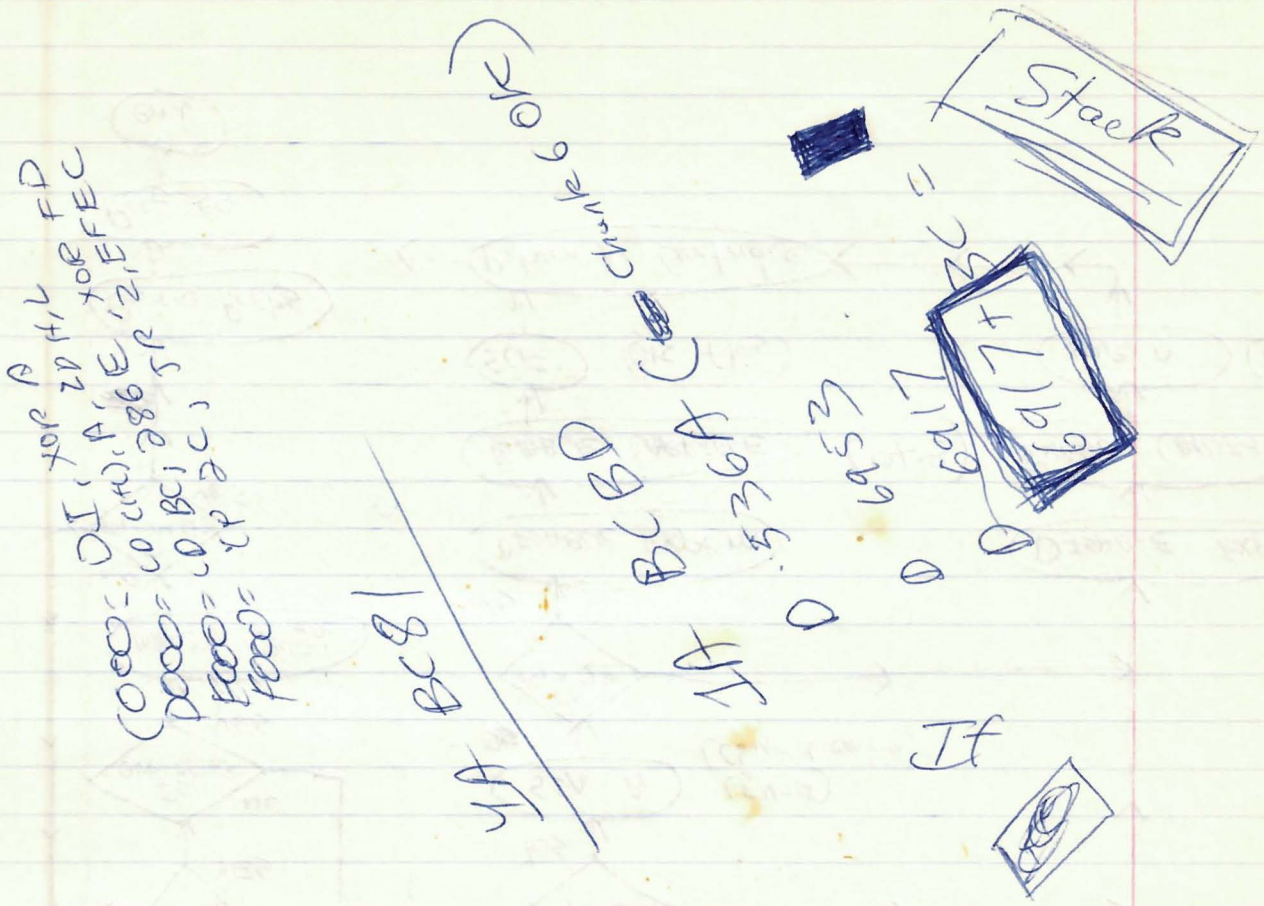
U8 → U15

FF = ØØ (Normal)
Ø1 = Second disply file
Ø2 = High Res Graphics
Ø4 = 64 Column color selection
Ø8

FF —

ØØ = Normal
Ø1 = Second display file
Ø2 = High-Res display
Ø4 = Undefined (Normal display + 64 column mode)
Ø8 =
16 = } Color selection for 64 Column mode
32 =
64 = INT enable (Ø) or disable (1)
128 = EXROM (1) or Cartridge (Ø) select (with F4)
255 =

(Can accept 191 (D7, D5-DØ set on)



Stack

(000: DI, XOR A
D000: LD (HL), A, LD H, L, FD
E000: LD BC, 286E, XOR 2, EFEC
F000: LP 2C, JP 2C,

1A BC81

1A BCBD (Chunk 6 OK)

D 536A

D 6953

D 6917

6917+BC =

If

EXROM Cartridge

8241 = 31 (Was 3Ø)
BA1D = Date for screen

11110000

EØ

128
64

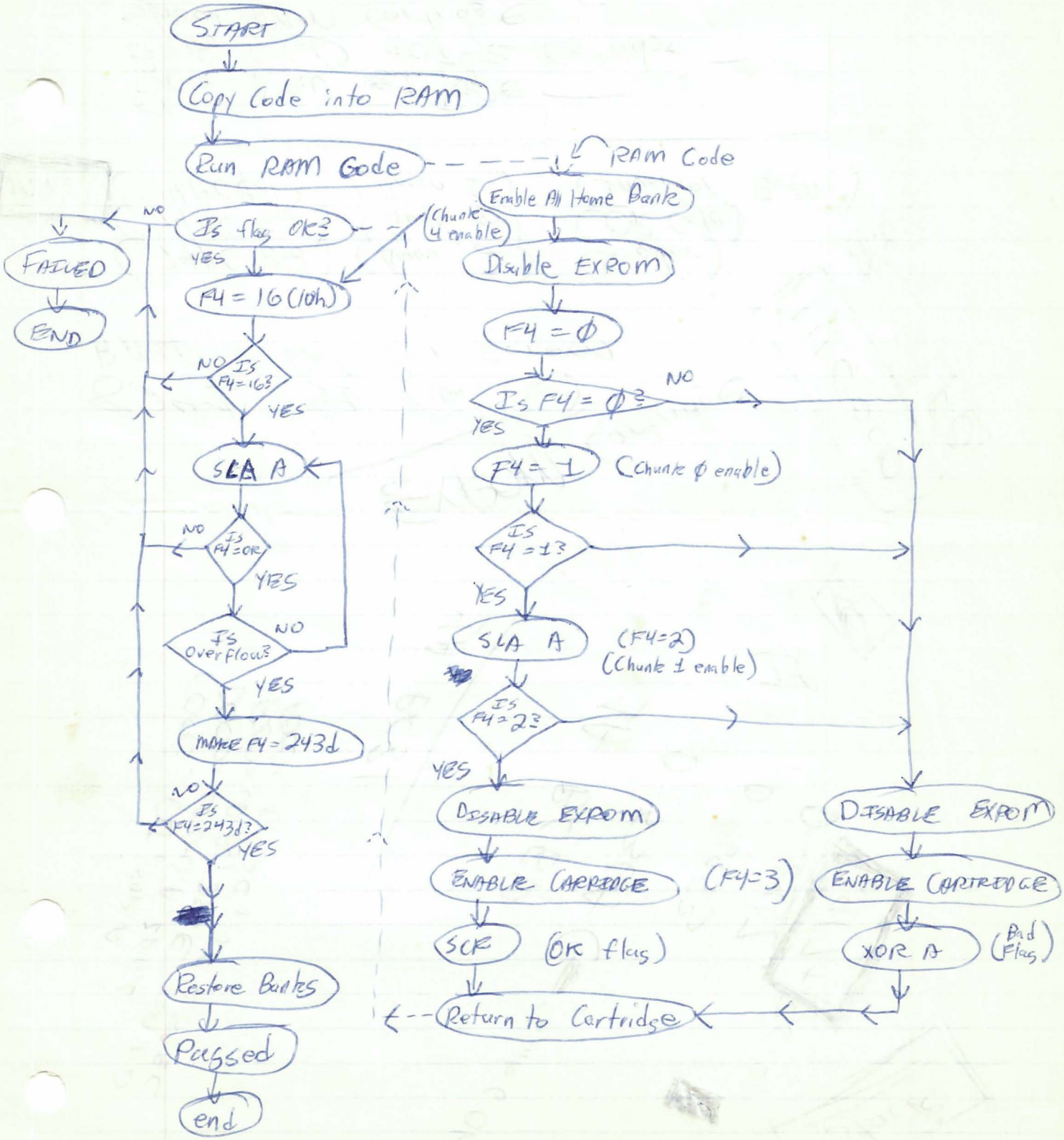
(74HC138) (Quan 2) (Decoder)
(74HC32) (Quan 2) (OR Gate)
MINE!! (74HCØ4) (Quan 1) (Inverter - Buffer)

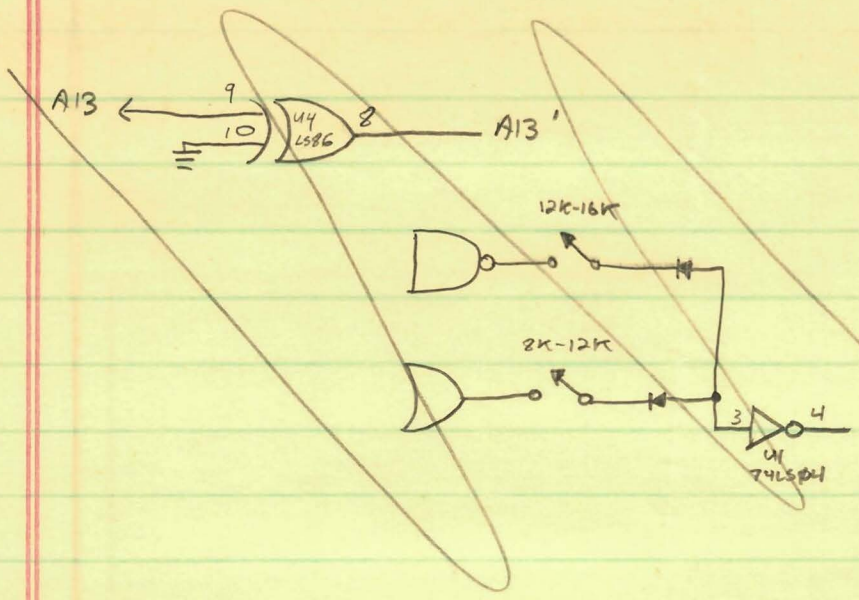
Bring New Test Tape —
Bring OLD HOT-Z Cartridge —
Show RAM Cartridge
Get New HOT-Z Cartridge —
Get New EXROM HOT-Z —

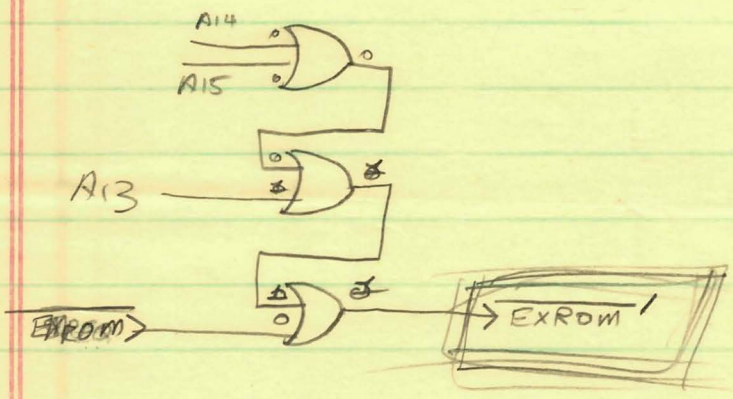
Federal Express

~~LALY~~ LOLLIE ← Number
Order is received

SCLO Test Flowchart







Write Protect /

Static RAMS (48K)

On

13E9 - ~~[illegible]~~ ~~[illegible]~~ 7086

New Length = 60h

1389 - F01E
138E - F023
13C8 - ~~[illegible]~~

709D

```
JP 8034
DI
LD HL,0000  }
ADD HL,SP   } SAVE OLD STACK
EX DE,HL    ; in DE
LD HL,6200  ; put stack at Timex Position (Chunk 3)
LD SP,HL    ; do it now
EX DE,HL    ; now old Position in HL again
  } rest of program
  }

JR NZ,93E2
LD A,C
OUT (F4),A
LD SP,HL    ; restore old stack
```

New EXROM HOT-Z (IS UPPER 8K USED?)
New CARTRIDGE HOT-Z (VERSION 2.45)
New TIMEX EXROM

CART

~~#~~: 1C51 = 7Ø9D

$\boxed{26}$

BD
BC
1A
ØØ

RAY KINGLSY 1-988-2096

8/12/85 - TALKED WITH RAY FOR 1:25:03:04 minutes
9/26/85 - 10:00 Minutes

[redacted] -BIN-
-ACS-

RAND UNIX 32776

DISB { 46994 - Chunk stack (Chunk) (
46995 - (BANK)

47209 - Chunk
47210 - BANK

330040 - CHUNK
33041 - BANK

8029h - 80 - EXROM
00 - DOCK

New EXROM [HOT-Z]
NEW CART. [Hoi-Z]

Fixes:

MAIN MENU: Change 'm' and 'w' to T/S 2068 style
: else OK

Home RAM Test: Fix Home RAM PASSED. Put a CR between 'Chips' and 'PASSED'.
: else, appears OK

CPU Test: Check entire Program

SCLD Test: LOCKS!!, check Program

HOME ROM Test:

Extension ROM Test

CASSETTE Test: Rewrite ROM for better Wait key

Keyboard Test: Make CAPS SHIFT & Space Bar 'beep'

TIMEX/SINCLAIR 2068 COLOR COMPUTER TEST SHEET

DATE:

SERIAL No. (P.C.B.)-

SERIAL No. (CASE)-

TESTS: √ = PASSED, X = Failed, + = Not tested

1) Home RAM (All 48K)-
2) CPU Test -
3) Home ROM (16K)-
4) EXROM (8K)-
5) CASSETTE TEST -
6) KEYBOARD -
7) COLORS, SYSTEM CLOCK, SYNC, MONITOR, TV -
8) BEEP AND SOUND CHIP -
9) JOYSTICK PORTS -
10) EXPANSION BUSS - (TESTED WITH A&J MICRODRIVE)

FINAL ASSEMBLY TESTS: DATE:

1) 6)
2) 7)
3) 8)
4) 9)
5) 10)

SPECIAL NOTES OR PROBLEMS:

SIGNATURE OF TECHNICIAN: _____ DATE:

TIMEX/SINCLAIR 2068 COMB COMPUTER TEST SHEET

DATE:

SERIAL No. (P.C.B.) -

SERIAL No. (CASE) -

TESTS: √ = PASSED, X = Failed, + = Not tested

1) Home RAM (All 48K) -
2) CPU Test -
3) Home ROM (16K) -
4) EXROM (8K) -
5) CASSETTE TEST -
6) KEYBOARD -
7) COLORS, SYSTEM CLOCK, SYNC, MONITOR, TV -
8) BEEP AND SOUND CHIP -
9) JOYSTICK PORTS -
10) EXPANSION BUSS - (TESTED WITH A&J MICRODRIVE)

FINAL ASSEMBLY TESTS: DATE:

1) 6)
2) 7)
3) 8)
4) 9)
5) 10)

SPECIAL NOTES OR PROBLEMS:

SIGNATURE OF TECHNICIAN: _____ DATE:

TIMEX/SINCLAIR 2068 COLOR COMPUTER TEST SHEET

DATE:

SERIAL No. (P.C.B.)-

SERIAL No. (CASE)-

TESTS: √ = PASSED, X = Failed, + = Not tested

1) Home RAM (All 48K)-
2) CPU Test-
3) Home ROM (16K)-
4) EXROM (8K)-
5) CASSETTE TEST-
6) KEYBOARD-
7) COLORS, SYSTEM CLOCK, SYNC, MONITOR, TV-
8) BEEP AND SOUND CHIP-
9) JOYSTICK PORTS-
10) EXPANSION BUSS- (TESTED WITH A&J MICRODRIVE)

FINAL ASSEMBLY TESTS: DATE:

1) 6)
2) 7)
3) 8)
4) 9)
5) 10)

SPECIAL NOTES OR PROBLEMS:

SIGNATURE OF TECHNICIAN: _____ DATE:

TIMEX/SINCLAIR 2068 COMB COMPUTER TEST SHEET

DATE:

SERIAL NO. (P.C.B.) -

SERIAL NO. (CASE) -

TESTS: √ = PASSED, X = FAILED, + = NOT TESTED

1) HOME RAM (ALL 48K) -
2) CPU TEST -
3) HOME ROM (16K) -
4) EXROM (8K) -
5) CASSETTE TEST -
6) KEYBOARD -
7) COLORS, SYSTEM CLOCK, SYNC, MONITOR, TV -
8) BEEP AND SOUND CHIP -
9) JOYSTICK PORTS -
10) EXPANSION BUSS - (TESTED WITH A&J MICRODRIVE)

FINAL ASSEMBLY TESTS: DATE:

1) 6)
2) 7)
3) 8)
4) 9)
5) 10)

SPECIAL NOTES OR PROBLEMS:

SIGNATURE OF TECHNICIAN: _____ DATE:

TIMEX/SINCLAIR 2068 COLOR COMPUTER TEST SHEET

DATE:

SERIAL No. (P.C.B.)-

SERIAL No. (CASE)-

TESTS: √ = PASSED, X = Failed, + = Not tested

1) Home RAM (All 48K)-
2) CPU Test-
3) Home ROM (16K)-
4) EXROM (8K)-
5) CASSETTE TEST-
6) KEYBOARD-
7) COLORS, SYSTEM CLOCK, SYNC, MONITOR, TV-
8) BEEP AND SOUND CHIP-
9) JOYSTICK PORTS-
10) EXPANSION BUSS- (TESTED WITH A&J MICRODRIVE)

FINAL ASSEMBLY TESTS: DATE:

1) 6)
2) 7)
3) 8)
4) 9)
5) 10)

SPECIAL NOTES OR PROBLEMS:

SIGNATURE OF TECHNICIAN: _____ DATE:

TIMEX/SINCLAIR 2068 COMB COMPUTER TEST SHEET

DATE:

SERIAL No. (P.C.B.)-

SERIAL No. (CASE)-

TESTS: √ = PASSED, X = Failed, + = Not tested

1) Home RAM (All 48K)-
2) CPU Test -
3) Home ROM (16K)-
4) EXROM (8K)-
5) CASSETTE TEST -
6) KEYBOARD -
7) COLORS, SYSTEM CLOCK, SYNC, MONITOR, TV -
8) BEEP AND SOUND CHIP -
9) JOYSTICK PORTS -
10) EXPANSION BUSS - (TESTED WITH A&J MICRODRIVE)

FINAL ASSEMBLY TESTS: DATE:

1) 6)
2) 7)
3) 8)
4) 9)
5) 10)

SPECIAL NOTES OR PROBLEMS:

SIGNATURE OF TECHNICIAN: _____ DATE: